

#2

520.3963200

JC978 U.S. PTO
09/784254
02/16/01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): FURUYA, ET AL.
Serial No.:
Filed: February 16, 2001
Title: METHOD AND APPARATUS FOR SYMMETRIC-KEY
ENCIPHERMENT
Group:

LETTER CLAIMING RIGHT OF PRIORITY

Honorable Commissioner of
Patents and Trademarks
Washington, D.C. 20231

520.39632X00

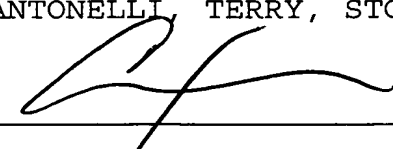
Sir:

Under the provisions of 35 USC 119 and 37 CFR 1.55, the applicant(s) hereby claim(s) the right of priority based on Japanese Patent Application No.(s) 2000-070994 filed March 9, 2000 and 2000-210690 filed July 6, 2000.

Certified copies of said Japanese Application are attached.

Respectfully submitted,

ANTONELLI, TERRY, STOUT & KRAUS, LLP



Carl I. Brundidge
Registration No. 29,621

CIB/mdt
Attachment
(703)312-6600

日 本 国 特 許 庁

PATENT OFFICE
JAPANESE GOVERNMENT

JC978 U.S. 1
09/784254
02/16/01

別紙添付の書類に記載されている事項は下記の出願書類に記載されて
る事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed
in this Office.

出 願 年 月 日
Date of Application:

2000年 3月 9日

願 番 号
Application Number:

特願2000-070994

願 人
Applicant(s):

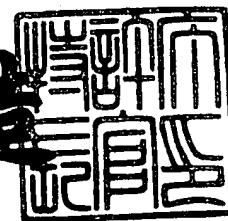
株式会社日立製作所

CERTIFIED COPY OF
PRIORITY DOCUMENT

2001年 1月12日

特許庁長官
Commissioner,
Patent Office

及 川 耕 造



出証番号 出証特2000-3111658

【書類名】 特許願

【整理番号】 K00000351

【提出日】 平成12年 3月 9日

【あて先】 特許庁長官殿

【国際特許分類】 G09C 1/00

【請求項の数】 30

【発明者】

【住所又は居所】 神奈川県川崎市麻生区王禅寺 1 0 9 9 番地 株式会社日
立製作所 システム開発研究所内

【氏名】 古屋 聡一

【発明者】

【住所又は居所】 神奈川県川崎市麻生区王禅寺 1 0 9 9 番地 株式会社日
立製作所 システム開発研究所内

【氏名】 宝木 和夫

【発明者】

【住所又は居所】 神奈川県横浜市戸塚区戸塚町 5 0 3 0 番地 株式会社日
立製作所 ソフトウェア事業部内

【氏名】 車谷 博之

【発明者】

【住所又は居所】 神奈川県川崎市麻生区王禅寺 1 0 9 9 番地 株式会社日
立製作所 システム開発研究所内

【氏名】 高橋 昌史

【発明者】

【住所又は居所】 神奈川県川崎市麻生区王禅寺 1 0 9 9 番地 株式会社日
立製作所 システム開発研究所内

【氏名】 宮崎 邦彦

【発明者】

【住所又は居所】 神奈川県川崎市麻生区王禅寺 1 0 9 9 番地 株式会社日
立製作所 システム開発研究所内

【氏名】 佐藤 尚宜

【特許出願人】

【識別番号】 000005108

【氏名又は名称】 株式会社日立製作所

【代理人】

【識別番号】 100075096

【弁理士】

【氏名又は名称】 作田 康夫

【手数料の表示】

【予納台帳番号】 013088

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 共通鍵暗号方法及び装置

【特許請求の範囲】

【請求項 1】

共通鍵暗号方法であって、

冗長データとメッセージとからなる平文を特定の長さで区切った複数の平文ブロックを生成し、

秘密鍵から、乱数列を生成し、前記乱数列から前記平文ブロックに対応する乱数ブロックを生成し、

前記平文ブロックと前記乱数ブロックとを演算して得た他の前記平文ブロックへのフィードバック値を出力し、

前記平文ブロックと、前記乱数ブロックと、他の平文ブロックの演算から得られるフィードバック値とを用いて暗号文ブロックを暗号演算する共通鍵暗号方法

。

【請求項 2】

請求項 1 記載の共通鍵暗号方法であって、

前記暗号演算は一つ以上の、その合計の長さは、前記暗号文ブロックより長い前記乱数ブロックを用いる共通鍵暗号方法。

【請求項 3】

請求項 2 記載の共通鍵暗号方法であって、

前記暗号演算として、前記平文ブロックを用いた 2 項演算や単項演算を決められた手順に従って 1 回以上行い、

得られた複数の暗号文ブロックを組み合わせ、暗号文として出力する共通鍵暗号方法。

【請求項 4】

請求項 2 記載の共通鍵暗号方法であって、

前記暗号演算を有限体上の乗算と加算により行う共通鍵暗号方法。

【請求項 5】

請求項 2 記載の共通鍵暗号方法であって、

前記暗号演算を巡回シフト演算と、算術乗算との組み合わせによって行う共通鍵暗号方法。

【請求項 6】

請求項 2 記載の共通鍵暗号方法であって、
擬似乱数生成手段を備え、前記秘密鍵から前記乱数列を生成する共通鍵暗号方法。

【請求項 7】

請求項 6 の共通鍵暗号方法であって、
前記メッセージを複数に分割し、
前記擬似乱数生成器を用いて、分割数分の乱数列を生成し、
前記分割したメッセージと前記乱数列のいずれかをそれぞれ異なる演算ユニットに割り当て、並列処理を行う共通鍵暗号方法。

【請求項 8】

共通鍵暗号の復号方法であって、
暗号文を特定の長さで区切った複数の暗号文ブロックを生成し、
秘密鍵から、乱数列を生成し、前記乱数列から前記暗号文ブロックに対応する乱数ブロックを生成し、
前記暗号文ブロックと前記乱数ブロックとを演算して得た他の前記暗号文ブロックへのフィードバック値を出力し、
前記暗号文ブロックと、前記乱数ブロックと、他の暗号文ブロックの演算から得られるフィードバック値と、を用いて平文ブロックを復号演算する共通鍵暗号の復号方法。

【請求項 9】

請求項 8 記載の共通鍵暗号の復号方法であって、
前記復号演算は一つ以上の、その合計の長さは、前記暗号文ブロックより長い前記乱数ブロックを用いる共通鍵暗号の復号方法。

【請求項 1 0】

請求項 9 記載の共通鍵暗号の復号方法であって、
前記平文ブロックを複数連結して平文を生成し、

前記平文に含まれる冗長データを抽出し、
前記冗長データを検査し、前記暗号文への改ざんの有無を検出する共通鍵暗号の復号化方法。

【請求項 1 1】

共通鍵暗号装置であって、
冗長データとメッセージとからなる平文を入力し、特定の長さで区切った複数の平文ブロックを生成する回路と、
秘密鍵を入力し、乱数列を生成し、前記乱数列から前記平文ブロックに対応する乱数ブロックを生成する回路と、
前記平文ブロックと前記乱数ブロックとを演算して得た他の前記平文ブロックへのフィードバック値を出力する回路と、
前記平文ブロックと、前記乱数ブロックと、他の平文ブロックと他の乱数ブロックとの演算によって得られるフィードバック値と、を用いて暗号文ブロックを暗号演算する回路と
を備える共通鍵暗号装置。

【請求項 1 2】

請求項 1 1 記載の共通鍵暗号装置であって、
前記暗号演算回路は、一つ以上の、その合計の長さは、前記暗号文ブロックより長い前記乱数ブロックを用いる、共通鍵暗号装置。

【請求項 1 3】

請求項 1 2 記載の共通鍵暗号装置であって、
前記暗号演算回路は、前記平文ブロックを用いた2項演算または単項演算を決められた手順に従って1回以上行う回路を含み、
得られた複数の暗号文ブロックを組み合わせ、暗号文として出力する回路を備える共通鍵暗号装置。

【請求項 1 4】

請求項 1 2 記載の共通鍵暗号装置であって、
前記暗号演算回路は、有限体上の乗算と加算を行う共通鍵暗号装置。

【請求項 1 5】

請求項 1 2 記載の共通鍵暗号装置であって、

前記暗号演算回路は、巡回シフト演算回路と算術乗算回路とを備える共通鍵暗号装置。

【請求項 1 6】

請求項 1 2 記載の共通鍵暗号装置であって、

擬似乱数生成器を備え、前記秘密鍵から前記乱数列を生成する共通鍵暗号装置

【請求項 1 7】

請求項 1 6 の共通鍵暗号装置であって、

前記メッセージを複数に分割する回路と、

前記擬似乱数生成器を用いて、分割数分の乱数列を生成する回路と、

複数の演算ユニットと、

前記分割したメッセージと前記乱数列のいずれかをそれぞれ異なる演算ユニットに割り当てる回路と、

を備える共通鍵暗号装置。

【請求項 1 8】

共通鍵暗号の復号装置であって、

暗号文を入力し、特定の長さで区切った複数の暗号文ブロックを生成する回路と、

秘密鍵を入力し、前記暗号文よりも長い乱数列を生成し、前記乱数列から前記暗号文ブロックに対応する乱数ブロックを生成する回路と、

前記暗号文ブロックと前記乱数ブロックとを演算して得た他の前記暗号文ブロックへのフィードバック値を出力する回路と、

前記暗号文ブロックと、前記乱数ブロックと、他の暗号文ブロックの演算から得られるフィードバック値と、を用いて平文ブロックを復号演算する回路とを備える共通鍵暗号の復号装置。

【請求項 1 9】

請求項 1 8 記載の共通鍵暗号の復号装置であって、

前記復号演算回路は、一つ以上の、その合計の長さは、長い前記乱数ブロックを用いる、共通鍵暗号の復号装置。

【請求項20】

請求項19記載の共通鍵暗号の復号装置であって、
前記平文ブロックを複数連結して平文を生成する回路と、
前記平文に含まれる冗長データを抽出する回路と、
前記冗長データを検査して前記暗号文への改ざんの有無を検出する回路とを備える共通鍵暗号の復号装置。

【請求項21】

コンピュータに読み込まれ、共通鍵暗号方法を実行させるプログラムを記憶した媒体であって、前記プログラムは、前記コンピュータに、
冗長データとメッセージとからなる平文を読み込ませ、前記平文を特定の長さで区切った複数の平文ブロックを生成させ、
秘密鍵を入力させ、乱数列を生成させ、前記乱数列から前記平文ブロックに対応する乱数ブロックを生成させ、
前記平文ブロックと前記乱数ブロックとを演算させて得た他の前記平文ブロックへのフィードバック値を出力させ、
前記平文ブロックと、前記乱数ブロックと、他の平文ブロックと他の乱数ブロックとの演算によって得られるフィードバック値と、を用いて暗号文ブロックを暗号演算させる
プログラムを記憶した媒体。

【請求項22】

請求項21記載のプログラムを記憶した媒体であって、
前記暗号演算として、一つ以上の、その合計の長さは、前記暗号文ブロックより長い前記乱数ブロックを用いさせる、
プログラムを記憶した媒体。

【請求項23】

請求項22記載のプログラムを記憶した媒体であって、
前記暗号演算として、前記平文ブロックを用いた2項演算または単項演算を決

められた手順に従って1回以上行なわせ、

得られた複数の暗号文ブロックを組み合わせ、暗号文として出力させるプログラムを記憶した媒体。

【請求項 2 4】

請求項 2 2 記載のプログラムを記憶した媒体であって、

前記暗号演算として、有限体上の乗算と加算を行わせるプログラムを記憶した媒体。

【請求項 2 5】

請求項 2 2 記載のプログラムを記憶した媒体であって、

前記暗号演算は、巡回シフト演算と算術乗算とを行わせるプログラムを記憶した媒体。

【請求項 2 6】

請求項 2 2 記載のプログラムを記憶した媒体であって、

擬似乱数生成を行わせ、前記秘密鍵から前記乱数列を生成させるプログラムを記憶した媒体。

【請求項 2 7】

請求項 2 6 のプログラムを記憶した媒体であって、

前記メッセージを複数に分割させ、

前記擬似乱数生成を、分割数分行わせ、

前記分割したメッセージと前記乱数列のいずれかをそれぞれ異なる演算ユニットに割り当てさせる、プログラムを記憶した媒体。

【請求項 2 8】

プログラムを記憶した媒体であって、

暗号文を入力し、特定の長さで区切った複数の暗号文ブロックを生成する回路と、

秘密鍵を入力し、前記暗号文よりも長い乱数列を生成し、前記乱数列から前記暗号文ブロックに対応する乱数ブロックを生成する回路と、

前記暗号文ブロックと前記乱数ブロックとを演算させて得た他の前記暗号文ブロックへのフィードバック値を出力させ、

前記暗号文ブロックと、前記乱数ブロックと、他の暗号文ブロックの演算から得られるフィードバック値と、を用いて平文ブロックを復号演算させるプログラムを記憶した媒体。

【請求項 2 9】

請求項 2 8 記載のプログラムを記憶した媒体であって、
前記復号演算として、一つ以上の、その合計の長さは、長い前記乱数ブロックを用いさせる、プログラムを記憶した媒体。

【請求項 3 0】

請求項 2 9 記載のプログラムを記憶した媒体であって、
前記平文ブロックを複数連結して平文を生成させ、
前記平文に含まれる冗長データを抽出させ、
前記冗長データを検査させて前記暗号文への改ざんの有無を検出させるプログラムを記憶した媒体。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本発明は、秘密情報のセキュリティを確保する技術に関する。

【0 0 0 2】

【従来の技術】

従来の暗号処理装置は、データを秘匿する目的のブロック暗号やストリーム暗号が提案されていた。ブロック暗号にはDES,IDEAなど非常にいろいろな方式が提案されている。DES,IDEAについては以下の文献で扱っている。

文献1:Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography, CRC Press, 1996, pp.250-259, pp.263-266。

【0 0 0 3】

ブロック暗号はECB,CBC,CFB,OFB,カウンタモードなどのブロック暗号操作モードにより全体の暗号処理の安全性や性質を議論するが、これまでに暗号化処理と改ざん検出を同時に行う操作モードは,iaPCBCモードが知られているだけで,残りのモードは改ざんの検出がそれ自身では不可能である。ブロック暗号の操作モー

ドについては以下の文献で扱っている。

文献2:Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, Inc., 1996, pp.189-209.

iaPCBCモードは以下の文献で扱っている。

文献3:Gligor, Donescu, "Integrity-Aware PCBC Encryption Schemes," Proceedings in Secure Protocol Workshop, Cambridge, 1999, to appear in Lecture Notes in Computer Science series, Springer-Verlag.

【 0 0 0 4 】

iaPCBCモードはブロック暗号を用いた操作モードであり、暗号化処理では並列処理が不可能、事前計算などが不可能、などの点で非常に高速な処理が要求される環境への実装が非常に困難であった。

これに対して、メッセージ認証子と呼ばれる改ざん検出のための暗号学的チェックサム(以下、MACという)を生成する方式が提案され、ブロック暗号の上記操作モードの暗号処理でも必要に応じてMAC生成処理を同時にかつまったく独立の機構として実装することで、暗号処理と改ざん検出が同時に可能となった。しかし、この場合にはまったく独立な暗号学的鍵を2度、すなわち、暗号化用と改ざん検出用、共有する必要があるという点、それから暗号化されるデータを2度処理、つまり、暗号化処理とMAC生成処理、にかけの必要があり、システムが複雑になったり、長いデータの処理に向かないなどの懸念があった。さらにブロック暗号の処理速度が現在の通信の速度に比べて低速であり、これらブロック暗号とMACの組み合わせ技術は、ギガビットやテラビット処理への応用が困難であった。MACについての記述は以下の文献にある。

文献4:Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography, CRC Press, 1996, pp.352-368.

【 0 0 0 5 】

これに対してストリーム暗号は、さまざま提案された暗号学的擬似乱数生成器を用いて暗号化を行うメカニズムで、擬似乱数生成器の安全性や性質とは無関係に改ざんの検出はそれ自身では不可能だった。ストリーム暗号、またはストリーム暗号に用いる擬似乱数生成器としてよく知られているものに、SEAL、非線形コンバ

イナを用いた線形フィードバックシフトレジスタ,非線形フィルタを用いた線形フィードバックシフトレジスタ,クロック制御方式の線形フィードバックシフトレジスタなどがある。SEALについては以下の文献で扱っている。

文献5:Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, Inc., 1996, pp.398-400。

【0006】

その他のフィードバックシフトレジスタに基づく方式については以下の文献で扱っている。

文献6:Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography, CRC Press, 1996, pp.203-212。

【0007】

ストリーム暗号とMACを組み合わせた技術により暗号処理と改ざん検出を同時に行えるほか,上記ブロック暗号の処理にくらべて,2倍~20倍の比率でストリーム暗号がより高速であるが,どのMAC生成方式についても,ブロック暗号とMACの組み合わせ同様,ふたつの異なる鍵の共有が必要であったり,ひとつのメッセージについて2度の処理を行う必要などがあった。より詳しくMAC生成方式を考えると,本来のストリーム暗号に,付带的に必要となるメカニズムや計算量が非常に大きい。例えば,HMAC,UMACなどのMAC生成方式では暗号学的に衝突なしで一方向性を保証している安全なハッシュ関数を必要であり,ストリーム暗号にさらに前述のハッシュ関数を実装する必要がある。HMACは前述の文献4,pp.355のExample 9.67で扱っている。またUMACは以下の文献で扱っている。

文献7:Black, Halevi, Krawczyk, Krovetz, Rogaway, "UMAC: Fast and Secure Message Authentication," Advances in Cryptology, - CRYPTO'99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, 1999。

【0008】

しかし,一般にSHA-1, MD5などのハッシュ関数は非常に複雑であり,実装も容易でない。これらのハッシュ関数については以下の文献で扱っている。

文献8:Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography, CRC Press, 1996, pp.347-349。

【 0 0 0 9 】

ハッシュ関数の安全性の面についても、ブロック暗号のそれに比べて議論が行われておらず、ユーザによってはハッシュ関数を信頼できず、組み込めないこともある。またMAC生成方式の中で擬似乱数生成器のみを用いる方式であるMMHのMAC生成では非常に少ない追加の資源、回路やプログラム、で暗号処理に改ざん検出の機能を付け加えることができるが、メッセージの長さに対してその2倍の長さの擬似乱数を必要とし、必要な乱数生成に時間がかかってしまう。MMHは以下の文献で扱っている。

文献9:Halevi, Krawczyk, "MMH: Software Message Authentication in the G bit/Second Rates," Fast Software Encryption, 4th International Workshop, FSE' 97, Lecture Notes in Computer Science, Vol. 1267, Springer-Verlag, 1997.

【 0 0 1 0 】

【発明が解決しようとする課題】

従来の暗号技術のほとんどは、復号化の際、改ざん検出をそれ自身ではすることができなかった。改ざん検出を行うとき、異なるふたつの鍵共有の必要性や、独立の処理、別の暗号学的要素関数の追加実装などが必要であった。

本発明は改ざん検出も復号化と同時に可能であり、またデータ秘匿、データ改ざんに対する安全性については、証明可能なものを提供することを目的とする。

処理速度の面についての課題として、ブロック暗号のこれまで知られた操作モードでは並列度や事前計算等の可能性がなく、高並列処理や高速処理には向いていないという点があった。

本発明は擬似乱数生成器の高速処理性能を生かしながら、事前計算や並列処理の利点をもつ、共通鍵暗号の方法を提供することを目的とする。

本発明は従来のブロック暗号よりも高速な処理が可能というだけではなく、資源が許される場合にはそれだけより高速な処理が可能であり、高い並列度を達成できるような高速処理に向けたものを提供することを目的とする。

本発明は非常に短いメッセージについても処理速度が落ちないような方式を提供することを目的とする。

本発明はストリーム暗号に非常に小さな追加の回路,またはプログラムで実装可能な方式を提供することを目的とする。

【 0 0 1 1 】

【課題を解決するための手段】

本発明では、擬似乱数列を鍵ストリームとして用いてブロック毎に処理を行い、かつ、改ざん検出を可能にする共通鍵暗号方法を提供する。

本発明の第一の態様は、平文P、鍵ストリームS、冗長性データ(以下、冗長性と略す)R、初期値Vから暗号文Cを得る暗号処理方法であって、鍵ストリームSの長さが暗号文Cより真に長い。

具体的には、冗長性の長さをbビット、平文の長さを

$L = n \times b + t$ ($0 \leq t < b$, $0 \leq n$) ビットとしたとき、平文の最後にビット文字列'0'を($(b-t) \bmod b$)個付加し、

冗長性Rを付加し、全体で

$L + ((b-t) \bmod b) + b$ ビットの文字列とする。これはbの倍数である。

これをbビットのブロックに分割し、 P_i ($1 \leq i \leq m$)とする。このとき必要な鍵ストリームの長さは $2mb$ ビットである。

この鍵ストリームは、暗号化装置側と復号化装置側とで、秘密裏に事前に共有しておくか、または、事前に共有した秘密鍵(例えば擬似乱数生成器の入力に相当する)から生成する。

上記長さの鍵ストリームを、2つのブロック列、 A_i, B_i ($1 \leq i \leq m$, それぞれbビット)に分割する。

フィードバック値の初期値 $F_0 = V$ として、以下により暗号文ブロック C_i を得る。(この初期値Vも事前に共有しておく。ただし、秘密にしておく必要はない)

$$F_i = P_i \wedge A_i, \quad C_i = (F_i * B_i) \wedge F_{i-1} \quad (1 \leq i \leq m).$$

得られた C_i をならべ、一つの文字列としたものを暗号文Cとして出力する。ここで、演算子「*」、「 \wedge 」は、それぞれ有限体 $F(2^b)$ 上での乗算、加算を示す。

【 0 0 1 2 】

対応する復号化は以下のとおりである。

暗号文C'の長さがbビットの倍数でない場合、非受理を出力する。もしbビットの

倍数であったなら、 b ビットのブロックに分割し、 $C'_i (1 \leq i \leq m')$ とする。

鍵ストリームブロック $A_i, B_i (1 \leq i \leq m')$ 、フィードバック値 $F'_0 = V$ をセットして、以下の処理を行う。

$$F'_i = (C'_i \wedge F'_{i-1}) / B_i, \quad P'_i = A_i \wedge F'_i \quad (1 \leq i \leq m').$$

得られた P'_i をならべ、一つの文字列としたものを復号化結果 P' として保存する。演算子「/」は有限体 $F(2^b)$ 上での除算を示す。

b ビット文字列 P'_m は、改ざんが起こらない場合、冗長性 R が復元されるべきである。また鍵を知らない攻撃者が改ざんを試みて、 P'_m が改ざん後にも変化しないような攻撃は $1/2^b$ 以上で成功しないことが保証されている。よってこれを根拠に、 b が十分に大きい (32以上) 場合には、 P'_m が冗長性 R と同じかどうかを検査することにより改ざんの検出が可能となる。

【0013】

本態様によれば、暗号文のいずれのブロックに改ざんが施されても、その影響が復号化時に最終ブロックまで伝播するという特徴がある。したがって、冗長データが変化しないような改ざんを、攻撃者が試みても防ぐことが可能になるという効果がある。

より具体的には、暗号化処理において、次のブロックのためのフィードバック値を生成、保存したのち、前のブロックの演算にて生成されたフィードバック値を用いて演算するものである。すなわち、暗号化処理において、 t 番目ブロックで生成される中間値を、生成される順に X_1, X_2, \dots, X_n とすると、次の $t+1$ 番目ブロックのためのフィードバック値 F_t となる X_i と、前のブロックからのフィードバック値 F_{t-1} が作用する中間値 X_j との引数 i, j について $i \leq j$ が成り立つ (必要条件)。

本態様によれば、悪意のある改ざんが改ざん検査をパスする確率が $1/2^b$ であるが、復号化において有限体での除算が生じ、また、平文の2倍の乱数を消費する。

第二の態様では安全性を少しゆるめて、さらに効率的な処理が可能な方法を示す。

第二の態様でも同様にメッセージ、冗長性の操作を同じように行う。冗長性付き平文が m ブロックのとき、必要な鍵ストリームの長さは $b \times (m+1)$ ビットであり、こ

れを A_i ($1 \leq i \leq m$) , B に分割する。

フィードバック値の初期値 $F_0 = V$ として、以下により暗号文ブロック C_i を得る。

$$F_i = P_i \cdot A_i, \quad C_i = (F_i * B) \cdot F_{i-1} \quad (1 \leq i \leq m).$$

得られた C_i をならべ、一つの文字列としたものを暗号文 C として出力する。

【 0 0 1 4 】

対応する復号化は以下のとおりである。暗号文 C' の長さが b ビットの倍数でない場合、非受理を出力する。もし b ビットの倍数であったなら、 b ビットのブロックに分割し、 C'_i ($1 \leq i \leq m'$)とする。暗号化と同様に鍵ストリームブロック A_i ($1 \leq i \leq m'$) , B 、フィードバック値 $F'_0 = V$ をセットして、以下の処理を行う。

$$F'_i = (C'_i \cdot F'_{i-1}) / B, \quad P'_i = A_i \cdot F'_i \quad (1 \leq i \leq m').$$

得られた P'_i から冗長性の部分を取り出し、決められた冗長性が得られたかどうかを検査する。もしそうであるなら、残りの P'_i の部分をメッセージとして出力し、そうでないなら、非受理を出力する。

ここで b ビット文字列 P'_m は、改ざんが起こらない場合、冗長性が復元されるべきである。

本態様によれば、 i 番目ブロックで使う複数の擬似乱数列(鍵ストリーム)のうちいずれかをブロック毎に変化させ、いずれかは共通に用いる。より具体的には、 i 番目のブロックで、暗号処理に与える擬似乱数列を、 A_i 、 B_i とすると、 A_i は毎ブロックごとに変化し、 B_i はすべてのブロックを通して変化しない。

この第二の態様では、鍵を知らない攻撃者が改ざん後に改ざん検出で検出されない確率は $(m-1)/2^b$ となる。一般的な改ざん検出確率は $1/2^{32}$ 以下であることが望ましく、 m は 2^{32} 程度が実際の実装でのデータ長の上限であるので、 $b=64$ 以上が望ましい。この場合、暗号化、復号化では有限体 $F(2^{64})$ 上の乗算を行うことになる。この演算はハードウェア実装ならば非常に高速かつ低コストで実現可能であるが、ソフトウェア実装の場合、次のような態様による高速化も考えられる。

【 0 0 1 5 】

第三の態様では、より長い冗長性についての方法を示す。まず、以降の処理を b ビット単位で行うのを前提として、冗長性を $b \times d$ ビットとする。第一、第二の態様と同様にメッセージ、冗長性の操作を同じように行い、 b ビットのメッセージと冗長

性によるブロック列, P_i ($1 \leq i \leq m, m \leq d$) が生成される。そして鍵ストリームの長さを $b \times (m+d)$ ビットとし, これを b ビットのブロック列 A_i ($1 \leq i \leq m$), B_i ($\neq 0, 1 \leq i \leq d$) に分割する。

フィードバック値の初期値 $F_0^{(i)} = V_i$ ($1 \leq i \leq d$) として, 以下により暗号文ブロック C_i を得る。

$$\begin{aligned} F_i^{(1)} &= P_i \wedge A_i, \\ F_i^{(j+1)} &= (F_i^{(j)} * B_j) \wedge F_{i-1}^{(j)} \quad (1 \leq j \leq d), \\ C_i &= F_i^{(d+1)} \quad (1 \leq i \leq m). \end{aligned}$$

得られた C_i をならべ, 一つの文字列としたものを暗号文 C として出力する。

【0016】

これに対応する復号化は以下のとおりである。暗号文 C' の長さが b ビットの倍数でない場合, 非受理を出力する。もし b ビットの倍数であったなら, b ビットのブロックに分割し, C'_i ($1 \leq i \leq m'$) とする。暗号化と同様に鍵ストリームブロック A_i ($1 \leq i \leq m'$), B_i ($\neq 0, 1 \leq i \leq d$), フィードバック値 $F_0^{(i)} = V_i$ ($1 \leq i \leq d$) をセットして, 以下の処理を行う。

$$\begin{aligned} F_i^{(d+1)} &= C'_i, \\ F_i^{(j)} &= (F_i^{(j+1)} \wedge F_{i-1}^{(j)}) / B_j \quad (1 \leq j \leq d), \\ P'_i &= A_i \wedge F_i^{(1)} \quad (1 \leq i \leq m). \end{aligned}$$

得られた P'_i から冗長性の部分を取り出し, 決められた冗長性が得られたかどうかを検査する。もしそうであるなら, 残りの P'_i の部分をメッセージとして出力し, そうでないなら, 非受理を出力する。

【0017】

第三の態様では, 冗長性の長さが $b \times d$ ビットでありながら, 暗号化や復号化に必要な処理は有限体 $F(2^b)$ 上の演算であり, これは有限体 $F(2^{b \times d})$ 上の乗算に比べて一般に $1/d^2$ 倍となる。しかし, 乗算の回数が d 倍になることから, この高速化方式は同じ冗長性の長さについて, 乗算の演算速度が約 $1/d$ の時間で処理可能となる。

第四の態様では, これまで挙げた第一から第三の態様の暗号処理で用いられる有限体 $F(2^b)$ 上の乗算を Feistel の 3 段構造に組み込んだものである。つまりこれまでの $A * B$ の演算を,

$$M_1 = A_L^{\wedge} (A_R * B_L), \quad M_2 = A_R^{\wedge} (M_1 * B_R), \quad M_3 = M_1^{\wedge} (M_2 * B_L),$$

により $M_3 || M_2$ を出力する関数に置き換える (B_L と B_R , A_L と A_R , M_2 と M_3 は, それぞれ入れ替わっても可)。これらの演算は自己逆演算となっており, 復号化の際, 同じ演算を用いることができる。

【 0 0 1 8 】

第五の態様では, メッセージの分割処理である。平文 P を送信者受信者間の決まりに従って t 個の文字列に分割し, それぞれを P_i ($1 \leq i \leq t$) とする。これまで述べた態様のひとつによりそれぞれ相異なる冗長性 R_i ($1 \leq i \leq t$) を用いて暗号化を行い暗号文 C_i を得る。またこれとは別に R_i の列のすべて, すなわち,

$(R_1 || R_2 || R_3 || \dots || R_t)$ を平文としてさらに送信者受信者間で共有された冗長性 R を用いて暗号化し, 暗号文 C_{t+1} を得る。これら, 暗号文ブロックの列の連結

$(C_1 || C_2 || C_3 || \dots || C_{t+1})$ を最終的な暗号文 C とする。

【 0 0 1 9 】

対応する復号化は, 暗号文を決められた方式に従って t 個の文字列に分割し, それぞれを独立に復号化する。どの復号化結果も非受理でなく, 冗長性 R_i すべてが, 冗長性を平文に見立てた部分の復号化結果に含まれていれば受理として, 冗長性の含まれる順番で復号化結果である平文を連結する。どれかの復号化で非受理ならば, 非受理とする。

【 0 0 2 0 】

第六の態様では, 上記五つの態様について, 有限体 $F(2^b)$ 上の乗算のかわりに, 有限体 $F(p)$ 上の乗算に基づいたものを考える (p はある整数 k によって 2^{k+1} と表すことができる素数である)。

具体的には $F(2^b)$ 上の乗算 $a * b$ の代わりとして, $F(p)$ 上の演算 $a * (b+1) + 1$ を行う。この演算は 2^b ビットレジスタ上で, 乗算 1 回, 加算 2 回, シフト 1 回で処理可能なため, 汎用的なプロセッサ上で乗算を高速に行うことができる。

上記演算は, $F(2^b)$ の乗算が, 排他的論理和 b 回, シフト b 回の計算を要し, 一般の素数 p における $F(p)$ の乗算が, 乗算 1 回と除算 1 回 (除算は, 通常, 加算やシフトの数十倍の時間がかかる) を要するのと比較して高速な処理が可能である。

【 0 0 2 1 】

本発明では擬似乱数を用いていることから、ユーザが最も信頼する暗号学的要素アルゴリズムは擬似乱数生成器として、ブロック暗号、ハッシュ関数、ストリーム暗号のどれかを用いることができ、安全性をユーザの選んだ暗号要素アルゴリズムに容易に帰着できる。

実装コストについては、ハッシュ関数の追加実装といった困難な追加実装を避けることができる。

【 0 0 2 2 】

【発明の実施の形態】

(実施例1)

図1はネットワーク10001によって接続されたコンピュータA10002、コンピュータB10003を含む、コンピュータA10002からコンピュータB10003への暗号通信を目的としたシステム構成を示すものである。コンピュータA10002は内部に演算装置(以下CPUという)10004、記憶装置(揮発性、不揮発性を問わない。以下RAMという)10005、ネットワークインターフェース10006を装備しており、外部にはコンピュータA10002をユーザが操作するためのディスプレイ10007とキーボード10008が接続されている。RAM10005には暗号化処理プログラムPROG1_10009、乱数生成処理プログラムPROG2_10010、コンピュータA10002とコンピュータB10003間のみで共有されている秘密情報である秘密鍵K10011、コンピュータA10002とコンピュータB10003間で共有されているデータである冗長性R10012と初期値V10013、それに暗号化してコンピュータB10003に送信したいデータ10014が保存されている。コンピュータB10003は内部にCPU10015、RAM10016、ネットワークインターフェース10017を装備しており、外部にはコンピュータB10003をユーザが操作するためのディスプレイ10018とキーボード10019が接続されている。RAM10016には復号化処理プログラムPROG3_10020、乱数生成処理プログラムPROG2_10021、秘密鍵K10011、冗長性R10012、それに初期値V10013が保存されている。

【 0 0 2 3 】

コンピュータA10002は、暗号化処理プログラムPROG1_10009を実行し、メッセージM10014の暗号文C10022を作成し、ネットワークインターフェース10006を通して

ネットワーク10001へ送信する。コンピュータB10003は、ネットワークインターフェース10017を通して受信したあと、復号化処理プログラムPROG3_10020を実行し、もし改ざんが検出されなければ復号化結果をRAM10016に保存する。

【0024】

各プログラムは、互いのコンピュータまたは他のコンピュータからネットワーク10001を介して、またはCD、FDなど可搬型記憶媒体を介してRAMに導入することができる。各プログラムは、各コンピュータのオペレーティングシステム(図示していない)の元で動作するように構成することも可能である。

暗号化処理プログラムPROG1_10009は、コンピュータA10002において、RAM10005から読み出されて、CPU10004により実行される。暗号化処理プログラムPROG1_10009は、サブルーチンとして乱数生成処理PROG2_10010を内部で呼び出し、入力秘密鍵K10011、冗長性R10012、初期値V10013、メッセージM10014に対して、暗号文C10022を出力する。

復号化処理プログラムPROG3_10020は、コンピュータB10003において、RAM10016から読み出されてCPU10015により実行される。復号化処理プログラムPROG3_10020は、サブルーチンとして乱数生成処理PROG2_10021を内部で呼び出し、入力秘密鍵K10011、冗長性R10012、初期値V10013、暗号文C10022に対して、メッセージ、または改ざん検知警告を出力する。

【0025】

暗号化処理プログラムPROG1_10009の処理の流れを説明する。

ステップ20002:データセットサブルーチン。初期値V、冗長性R、秘密鍵Kの入力を待つ。

ステップ20003:平文準備サブルーチン。平文の入力を待ち、平文が与えられたあと決められたパディングを行い、冗長性を付加して、最後に64ビットごとに区切って平文ブロックの列 P_i ($1 \leq i \leq n$)を出力する。

ステップ20004:乱数生成サブルーチン。秘密鍵Kから、擬似乱数列 A_i, B_i ($1 \leq i \leq n$)を出力する。

ステップ20005:暗号化サブルーチン。擬似乱数列 A_i, B_i 、平文ブロック列 P_i ($1 \leq i \leq n$)、初期値Vを使って、暗号文ブロック C_i ($1 \leq i \leq n$)を出力する。

ステップ20006:ステップ20005で得られた暗号文ブロック C_i ($1 \leq i \leq n$)を順にビット連結し,暗号文Cとして出力する。

【 0 0 2 6 】

平文準備サブルーチンの処理を図2を用いて説明する。

【 0 0 2 7 】

ステップ20202:暗号処理に用いるメッセージMの入力を待つ。メッセージMは、キーボード10008から入力されたり、RAM内に保存されていたり、他の記憶媒体から導入されたりする。

ステップ20203:メッセージの長さを示すデータをパディング。メッセージMの先頭にメッセージMのビット長を表す64ビット二進数データを付加する。

ステップ20204:メッセージ長をそろえるパディング。後の暗号処理のためパディング後のデータを64ビットの整数倍にする。メッセージMの長さをLビットとすると $64 - L \pmod{64}$ 個の0をステップ20203で長さデータを付加されたメッセージの末尾にパディングする。

ステップ20205:冗長性データのパディング。64ビットデータとして与えられる冗長性Rをさらに末尾にパディングする。

ステップ20206:メッセージデータの平文ブロックへ分割。ステップ20205の結果得られたデータを64ビットのブロックに区切り順に P_1, P_2, \dots, P_n とする。

【 0 0 2 8 】

乱数生成サブルーチンの処理を図3を用いて説明する。

ステップ20302:必要パラメータの入力。パディング後のメッセージブロック数nと,秘密鍵Kを得る。

ステップ20303:擬似乱数列Aの生成。乱数生成処理プログラムPROG2を呼び出し,長さ $64n$ ビットの擬似乱数列を生成,出力をAとする。

ステップ20304:乱数列Aをブロックに分割。擬似乱数列Aを64ビットごとに区切り順に A_1, A_2, \dots, A_n とする。

ステップ20305:カウンタiの初期化。カウンタ $i=1$ とする。

ステップ20306:乱数 B_i の生成。秘密鍵Kを使ってPROG2を実行することにより64ビットの乱数 B_i を生成する。

ステップ20307: ステップ20306で生成された B_i が0ならば、ステップ20306に戻る。

ステップ20308: $i=n$ ならばステップ20310を実行。

ステップ20309: カウンタ i をインクリメントしステップ20306へ戻る。

【 0 0 2 9 】

暗号化サブルーチンの処理を図4を用いて説明する。

ステップ20402: 初期値セット。 $F_0=V$ とする。

ステップ20403: カウンタ初期化。 $i=1$ とする。

ステップ20404: フィードバック値 F_i 計算。 $F_i=P_i \wedge A_i$ とする。

ステップ20405: 暗号文ブロック C_i 計算。 $C_i=(F_i * B_i) \wedge F_{i-1}$ とする。

ステップ20406: $i=n$ ならばステップ20408を実行。

ステップ20407: カウンタ i をインクリメントしステップ20404へ戻る。

【 0 0 3 0 】

復号化処理プログラムPROG3_10020の処理の流れを図5を用いて説明する。

ステップ20502: データセットサブルーチン。初期値 V , 冗長性 R , 秘密鍵 K の入力を待つ。

ステップ20503: 暗号文準備サブルーチン。暗号文 C' の入力を待ち, 暗号文 C' が与えられたあと64ビットごとに区切って暗号文ブロックの列 C'_i ($1 \leq i \leq n$)を出力する。

ステップ20504: 乱数生成サブルーチン。秘密鍵 K から, 擬似乱数列 A_i, B_i ($1 \leq i \leq n$)を出力する。

ステップ20505: 復号化サブルーチン。擬似乱数列 A_i, B_i , 暗号文ブロック列 C'_i ($1 \leq i \leq n$), 初期値 V を使って, 平文ブロック P'_i ($1 \leq i \leq n$)を出力する。

ステップ20506: 平文切り出しサブルーチン。平文ブロックの列 P'_i を3つのデータ列, L', M', Z' に分割する。

ステップ20507: 冗長性切り出しサブルーチン。 Z' を R' と T' に分割する。

ステップ20508: $T=0$ かつ $R'=R$ ならばステップ20510に進む。

ステップ20509: 拒否 (非受理) を出力。ステップ20511に進む。

ステップ20510: M' をRAMへ格納する。

ステップ20509,20510では、復号化処理プログラムは、結果(受理/非受理あるいは復号結果)をディスプレイ10018に出力して、ユーザに結果を通知する。

【 0 0 3 1 】

暗号文準備サブルーチンの処理を図6を用いて説明する。

ステップ20602:暗号文 C' の入力を待つ。

ステップ20603:暗号文 C' を64ビットごとに区切り,順に C'_1, C'_2, \dots, C'_n とする
復号化サブルーチンの処理を図7を用いて説明する。

【 0 0 3 2 】

ステップ20702:初期値セット。 $F'_0 = V$ とする。

ステップ20703:カウンタ初期化。 $i=1$ とする。

ステップ20704:フィードバック値 F'_i 計算。 $F'_i = (C'_i \wedge F'_{i-1}) / B_i$ とする。

ステップ20705:平文ブロック P'_i 計算。 $P'_i = F'_i \wedge A_i$ とする。

ステップ20706: $i=n$ ならばステップ20708を実行。

ステップ20707:カウンタ i をインクリメントしステップ20704へ戻る。

【 0 0 3 3 】

平文切り出しサブルーチンの処理を図8を用いて説明する。

ステップ20802: L' を最初の64ビットの平文ブロックとする。

ステップ20803:復号文ブロックのうち P'_2 最上位ビットから L' ビットのデータを M' とする。

ステップ20804: M' を切り取った残りのデータを Z' とする。

【 0 0 3 4 】

冗長性切出しサブルーチンの処理を図9を用いて説明する。

ステップ20902: R' を Z' の下位64ビットのデータとする。

ステップ20903: R' を切り取った後の残りのデータを T' とする。

【 0 0 3 5 】

図10は、暗号化処理の説明図である。○印の中に+が入った記号は64ビット幅の二つのデータによる排他的論理和演算を表し、○印の中に×が入った記号は64ビット幅の二つのデータによる有限体 $F(2^{64})$ 上での乗算を示す。

メッセージM20931に長さ20930,適当なパディング20932,冗長性R20933をそれぞれ

れ付加し、平文P20934を生成する。

これを64ビットに分割したものをそれぞれ P_1 -20935, P_2 -20936, P_3 -20937, P_n -20938とする。

P_1 -20935は A_1 -20940と排他論理和を取り、フィードバック値 F_1 -20941とし、 B_1 -20942と有限体上の乗算を行い、その結果と初期値 F_0 -20939の排他論理和により暗号文ブロック C_1 -20943を得る。

P_2 -20936は A_2 -20946と排他論理和を取り、フィードバック値 F_2 -20945とし、 B_2 -20946と有限体上の乗算を行い、その結果とフィードバック値 F_1 -20941の排他論理和により暗号文ブロック C_2 -20947を得る。

同様にして P_n -20938まで処理を行い、暗号文ブロック C_1 -20943, C_2 -20947, C_3 -20951,..., C_n -20955を得たあと、これらをこの順で連結し、暗号文 C -20956を得る。

【 0 0 3 6 】

図11は、復号化処理の説明図である。○印の中に/が入った記号は64ビット幅の二つのデータによる有限体 $F(2^{64})$ 上での除算を示す。該記号の上から入るデータが割られる数、該記号の左から入るデータが割る数である。

暗号文 C' -20960を64ビットのブロックに分割し、 C'_1 -20962, C'_2 -20963, C'_3 -20964,..., C'_n -20965とする。

C'_1 は初期値 F'_0 -20961と排他的論理和をとり、その結果を B_1 -20966で割る。この結果をフィードバック値 F'_1 -20967とする。平文ブロック P'_1 -20969はフィードバック値 F'_1 -20967と A_1 -20968との排他論理和により得られる。

C'_2 -20963, C'_3 -20964,..., C'_n -20965も同様に処理を行い、得られた平文ブロック P'_1 -20969, P'_2 -20972, P'_3 -20977,..., P'_n -20981を連結し平文 P' -20982とする。これを L' -20983, M' -20984, Z' -209985に分割後、さらに Z' -209985を T' -20988と R' -20989に分割し、冗長性のチェックを行う。

【 0 0 3 7 】

実施例1では、メッセージの長さの約2倍の長さだけの擬似乱数を暗号処理に使用した。擬似乱数はブロック暗号に比較して高速ではあるものの、この暗号処理で最も計算量のかかる処理であり、消費する乱数の数を減らすことが望ましい。

(実施例2)

以下に示す実施例2では、別の関数を使うことにより、より少ない擬似乱数の消費で行え、また復号化の際の割る数も変化せず、逆数の事前計算によりほぼ乗算と同じ速度で達成できるため、極めて効率的な処理が可能となる。

実施例2では、暗号化処理PROG1、復号化処理PROG3の代わりに暗号化処理PROG1A、復号化処理PROG3Aをそれぞれ使う。

暗号化処理PROG1Aを示す。PROG1Aは、図1のPROG1_10009において、乱数生成サブルーチン20004と暗号化サブルーチン20005をそれぞれ乱数生成2サブルーチン21004と暗号化2サブルーチン21005に置き換えたものである。

【 0 0 3 8 】

乱数生成2サブルーチン21004の処理を図12を用いて説明する。

【 0 0 3 9 】

ステップ21102:必要パラメータの入力。パディング後のメッセージブロック数 n と、秘密鍵 K を得る。

ステップ21103:擬似乱数列 A の生成。乱数生成処理プログラムPROG2を呼び出し、長さ $64n$ ビットの擬似乱数列を生成、出力を A とする。

ステップ21104:乱数列 A をブロックに分割。擬似乱数列 A を64ビットごとに区切り順に A_1, A_2, \dots, A_n とする。

ステップ21105:乱数 B の生成。秘密鍵 K を使ってPROG2を実行することにより64ビットの乱数 B を生成する。

ステップ21106:ステップ21105で生成された B が0ならば、ステップ21105に戻る。

【 0 0 4 0 】

暗号化2サブルーチン21005の処理を図13を用いて説明する。

ステップ21202:初期値セット。 $F_0 = V$ とする。

ステップ21203:カウンタ初期化。 $i = 1$ とする。

ステップ21204:フィードバック値 F_i 計算。 $F_i = P_i \wedge A_i$ とする。

ステップ21205:暗号文ブロック C_i 計算。 $C_i = (F_i * B) \wedge F_{i-1}$ とする。

ステップ21206:もし、 $i = n$ ならばステップ21208を実行。

ステップ21207:カウンタ i をインクリメントしステップ21204へ戻る。

【 0 0 4 1 】

PROG1Aと対となる、復号化処理PROG3Aの処理の流れを図14を用いて説明する。PROG3Aは、PROG3_10020において、乱数生成サブルーチン20504と復号化サブルーチン20505をそれぞれ乱数生成2サブルーチン21304と復号化2サブルーチン21305に置き換えたものである。

【 0 0 4 2 】

ステップ21302:データセットサブルーチン。初期値V,冗長性R,秘密鍵Kの入力を待つ。

ステップ21303:暗号文準備サブルーチン。暗号文C'の入力を待ち,暗号文C'が与えられたあと64ビットごとに区切って暗号文ブロックの列C'_i (1≤i≤n)を出力する。

ステップ21304:乱数生成サブルーチン。秘密鍵Kから,擬似乱数列A_i (1≤i≤n), Bを出力する。

ステップ21305: 復号化サブルーチン。擬似乱数列A_i, B, 暗号文ブロック列C'_i (1≤i≤n), 初期値Vを使って, 平文ブロックP'_i (1≤i≤n)を出力する。

ステップ21306:平文切り出しサブルーチン。平文ブロックの列P'_iを3つのデータ列,L', M', Z' に分割する。

ステップ21307:冗長性切り出しサブルーチン。Z' をR' とT' に分割する。

ステップ21308:T=0かつR'=Rならばステップ21310に進む。

ステップ21309:拒否を出力。ステップ21311に進む。

ステップ21310:M' をRAMへ格納する。

【 0 0 4 3 】

図14の復号化2サブルーチン21305の処理を図15を用いて説明する。

ステップ21402:初期値セット。F'₀=Vとする。

ステップ21403:1/Bを事前計算しておく。

ステップ21404:カウンタ初期化。i=1とする。

ステップ21405:フィードバック値F'_i計算。F'_i=(C'_i ^ F'_{i-1})*(1/B)とする。

ステップ21406:平文ブロックP'_i計算。P'_i=F'_i ^ A_iとする。

ステップ21407:i=nならばステップ21409を実行。

ステップ21408:カウンタ i をインクリメントしステップ21405へ戻る。

【 0 0 4 4 】

図16は、上述の高速化手法で用いた暗号化処理の説明図である。

メッセージ M_{21421} に長さ21420,適当なパディング21422,冗長性 R_{21423} をそれぞれ付加し,平文 P_{21424} を生成する。

これを64ビットに分割したものをそれぞれ $P_{1_21425}, P_{2_21426}, P_{3_21427}, P_{n_21428}$ とする。

P_{1_21425} は A_{1_21431} と排他論理和を取り,フィードバック値 F_{1_21432} とし, B_{21429} と有限体上の乗算を行い,その結果と初期値 F_{0_21430} の排他論理和により暗号文ブロック C_{1_21433} を得る。

P_{2_21426} は A_{2_21434} と排他論理和を取り,フィードバック値 F_{2_21435} とし, B_{21429} と有限体上の乗算を行い,その結果とフィードバック値 F_{1_21432} の排他論理和により暗号文ブロック C_{2_21436} を得る。

同様にして P_{n_21428} まで処理を行い,暗号文ブロック $C_{1_21433}, C_{2_21436}, C_{3_21439}, \dots, C_{n_21442}$ を得たあと,これらをこの順で連結し,暗号文 C_{21443} を得る。

【 0 0 4 5 】

図17は,対応する復号化処理の説明図である。

暗号文 C'_{21450} は64ビットのブロックに分割し, $C'_{1_21453}, C'_{2_21454}, C'_{3_21455}, \dots, C'_{n_21456}$ とする。

C'_{1} は初期値 F'_{0_21451} と排他的論理和をとり,その結果と $1/B_{21452}$ の乗算を計算する。この結果をフィードバック値 F'_{1_21457} とする。平文ブロック P'_{1_21459} はフィードバック値 F'_{1_21457} と A_{1_21458} との排他論理和により得られる。

$C'_{2_21454}, C'_{3_21455}, \dots, C'_{n_21456}$ も同様に処理を行い,得られた平文ブロック $P'_{1_21459}, P'_{2_21462}, P'_{3_21465}, \dots, P'_{n_21468}$ を連結し平文 P'_{21476} とする。これを $L'_{21469}, M'_{21470}, Z'_{21471}$ に分割後,さらに Z'_{21471} を T'_{21474} と R'_{21475} に分割し,冗長性のチェックを行う。

実施例2では,64ビットの冗長性を用いるために有限体 $F(2^{64})$ 上の加算,乗算を用いている。

(実施例3)

以下に示す実施例3では、別の高速な関数を使うことにより同じ安全性かつより高速な処理が可能となる。観点を変え、これまでと同じ $F(2^{64})$ 上の演算を使った場合には($F(2^{128})$ の演算相当の)より高い安全性を提供することができる。

高速化の観点では、一般に $F(2^{64})$ 上の乗算は $F(2^{32})$ 上のそれに比べ4倍の計算量がかかるが、実施例3では1/4の計算量の演算を2回繰り返すので事実上2倍の高速処理が可能となる。

安全性の補強という観点では、 $F(2^{64})$ 上の演算と64ビットのフィードバック値を2度用いることで、改ざんの成功確率を、上記手法の 2^{-64} から 2^{-128} へより小さくすることができる。

この高速化手法として暗号化処理PROG1の代わりに同じく暗号化処理PROG1B、復号化処理PROG3の代わりに同じく復号化処理PROG3Bをそれぞれ使う装置を考える。

【 0 0 4 6 】

暗号化処理PROG1Bは、図1に示すPROG1_1009において、乱数生成サブルーチン(ステップ20004)と暗号化サブルーチン(ステップ20005)をそれぞれ乱数生成3サブルーチン21504と暗号化3サブルーチン21505に置き換えたものである。暗号化処理PROG1Bの処理の流れを図18を用いて説明する。

ステップ21502:データセットサブルーチン。初期値V,冗長性R,秘密鍵Kの入力を待つ。

ステップ21503:平文準備サブルーチン。平文の入力を待ち、平文が与えられたあと決められたパディングを行い、冗長性を付加して、最後に32ビットごとに区切って平文ブロックの列 P_i ($1 \leq i \leq n$)を出力する。

ステップ21504:乱数生成3サブルーチン。秘密鍵Kから、擬似乱数列 A_i ($1 \leq i \leq n$), Ba , Bb を出力する。

ステップ21505:暗号化3サブルーチン。擬似乱数列 A_i , Ba , Bb , 平文ブロック列 P_i ($1 \leq i \leq n$), 初期値Vを使って、暗号文ブロック C_i ($1 \leq i \leq n$)を出力する。

ステップ21506:ステップ21505で得られた暗号文ブロック C_i ($1 \leq i \leq n$)を順にビット連結し、暗号文Cとして出力する。

【 0 0 4 7 】

乱数生成3サブルーチン21504の処理を図19を用いて説明する。

ステップ21602:必要パラメータの入力。パディング後のメッセージブロック数 n と,秘密鍵 K を得る。

ステップ21603:擬似乱数列 A の生成。乱数生成処理プログラムPROG2を呼び出し,長さ $32n$ ビットの擬似乱数列を生成,出力を A とする。

ステップ21604:乱数列 A をブロックに分割。擬似乱数列 A を32ビットごとに区切り順に A_1, A_2, \dots, A_n とする。

ステップ21605:乱数 Ba の生成。秘密鍵 K を使ってPROG2を実行することにより32ビットの乱数 Ba を生成する。

ステップ21606:ステップ21605で生成された Ba が0ならば,ステップ21605に戻る。

ステップ21607:乱数 Bb の生成。秘密鍵 K を使ってPROG2を実行することにより32ビットの乱数 Bb を生成する。

ステップ21608:ステップ21607で生成された Bb が0ならば,ステップ21607に戻る。

【 0 0 4 8 】

暗号化3サブルーチン21505の処理を図20を用いて説明する。記号「 $*$ 」,「 $^$ 」はそれぞれ有限体 $F(2^{32})$ 上での乗算,加算を表す。

ステップ21702:初期値セット。 $FA_0=FB_0=V$ とする。

ステップ21703:カウンタ初期化。 $i=1$ とする。

ステップ21704:フィードバック値 FA_i 計算。 $FA_i=P_i^*A_i$ とする。

ステップ21705:フィードバック値 FB_i 計算。 $FB_i=(FA_i*Ba)^*FA_{i-1}$ とする。

ステップ21706:暗号文ブロック C_i 計算。 $C_i=(FB_i*Bb)^*FB_{i-1}$ とする。

ステップ21707: $i=n$ ならばステップ21709を実行。

ステップ21708:カウンタ i をインクリメントしステップ21704へ戻る。

【 0 0 4 9 】

復号化処理PROG3Bの処理の流れを図21を用いて説明する。PROG3Bは,PROG3_10020において,乱数生成サブルーチン20504と復号化サブルーチン20505をそれぞれ

乱数生成3サブルーチン21804と復号化3サブルーチン21805に置き換えたものである。

ステップ21802:データセットサブルーチン。初期値V,冗長性R,秘密鍵Kの入力を待つ。

ステップ21803:暗号文準備サブルーチン。暗号文C'の入力を待ち,暗号文C'が与えられたあと32ビットごとに区切って暗号文ブロックの列 C'_i ($1 \leq i \leq n$)を出力する。

ステップ21804:乱数生成サブルーチン。秘密鍵Kから,擬似乱数列 A_i ($1 \leq i \leq n$), Ba, Bbを出力する。

ステップ21805:復号化サブルーチン。擬似乱数列 A_i , Ba, Bb, 暗号文ブロック列 C'_i ($1 \leq i \leq n$), 初期値Vを使って, 平文ブロック P'_i ($1 \leq i \leq n$)を出力する。

ステップ21806:平文切り出しサブルーチン。平文ブロックの列 P'_i を3つのデータ列, L', M', Z' に分割する。

ステップ21807:冗長性切り出しサブルーチン。Z' をR' とT' に分割する。

ステップ21808: $T=0$ かつ $R'=R$ ならばステップ21810に進む。

ステップ21809:拒否を出力。ステップ21811に進む。

ステップ21810:M' をRAMへ格納する。

【0050】

図21の復号化3サブルーチン21805の処理を図22を用いて説明する。記号「/」は有限体 $F(2^{32})$ 上での除算を表す。

ステップ21902:初期値セット。 $FA'_0 = FB'_0 = V$ とする。

ステップ21903: $1/Ba, 1/Bb$ を事前計算しておく。

ステップ21904:カウンタ初期化。 $i=1$ とする。

ステップ21905:フィードバック値 FB'_i 計算。 $FB'_i = (C'_i \wedge FB'_{i-1}) * (1/Bb)$ とする。

ステップ21906:フィードバック値 FA'_i 計算。 $FA'_i = (FB'_i \wedge FA'_{i-1}) * (1/Ba)$ とする。

ステップ21907:平文ブロック P'_i 計算。 $P'_i = FA'_i \wedge A_i$ とする。

ステップ21908: $i=n$ ならばステップ21910を実行。

ステップ21909:カウンタ i をインクリメントしステップ21905へ戻る。

【 0 0 5 1 】

図23は、上述の高速化手法で用いた暗号化処理の説明図である。

メッセージ M_{21921} に長さ L_{21920} ,適当なパディング 21922 ,冗長性 R_{21923} をそれぞれ付加し,平文 P_{21924} を生成する。

これを64ビットに分割したものをそれぞれ $P_1_{21925}, P_2_{21926}, P_3_{21927}, P_n_{21928}$ とする。

P_1_{21925} は A_1_{21933} と排他論理和を取り,フィードバック値 FA_1_{21934} とし, Ba_{21929} と有限体上の乗算を行い,その結果と初期値 FA_0_{21930} の排他論理和により,フィードバック値 FB_1_{21939} を得る。これを Bb_{21931} と有限体上の乗算を行い,その結果と初期値 FB_0_{21932} の排他論理和により暗号文ブロック C_1_{21936} を得る。

P_2_{21926} は A_2_{21937} と排他論理和を取り,フィードバック値 FA_2_{21942} とし, Ba_{21929} と有限体上の乗算を行い,その結果とフィードバック値 FA_1_{21934} の排他論理和により,フィードバック値 FB_2_{21943} を得る。これを Bb_{21931} と有限体上の乗算を行い,その結果とフィードバック値 FB_1_{21939} の排他論理和により暗号文ブロック C_2_{21940} を得る。

同様にして P_n_{21928} まで処理を行い,暗号文ブロック $C_1_{21936}, C_2_{21940}, C_3_{21944}, \dots, C_n_{21950}$ を得たあと,これらをこの順で連結し,暗号文 C_{21951} を得る。

【 0 0 5 2 】

図24は、対応する復号化処理の説明図である。

暗号文 C'_{21960} は64ビットのブロックに分割し, $C'_1_{21961}, C'_2_{21962}, C'_3_{21963}, \dots, C'_n_{21964}$ とする。

C'_1 は初期値 FB'_0_{21965} と排他的論理和をとり,その結果と $1/Bb_{21966}$ の乗算を計算する。この結果をフィードバック値 FB'_1_{21969} とする。さらに,初期値 FA'_0_{21967} と排他的論理和をとり,その結果と $1/Ba_{21968}$ の乗算を計算する。平文ブロック P'_1_{21972} はフィードバック値 FA'_1_{21970} と A_1_{21971} との排他論理和により得られる。

$C'_2_{21962}, C'_3_{21976}, \dots, C'_n_{21964}$ も同様に処理を行い,得られた平文ブロック $P'_1_{21972}, P'_2_{21976}, P'_3_{21980}, \dots, P'_n_{21985}$ を連結し平文 P'_{21986} とする。

これをL'_21987,M'_21988,Z'_21989に分割後、さらにZ'_21989をT'_21992とR'_21993に分割し、冗長性のチェックを行う。

【 0 0 5 3 】

(実施例4)

上記実施例1から3は並列計算を考えない単一プロセッサによる処理の実施形態であった。実施例4において、本発明が並列処理への応用も簡単であることを示す。

実施例4のシステム構成が、図1の構成と異なるのは、コンピュータA10002については、CPU10004の代わりにCPU1_30004,CPU2_30005を装備しており、RAM10005には、図1の構成に加えて並列暗号処理プログラムPROG4_30016が保存されている点である。コンピュータB10003については、CPU10015の代わりにCPU1_30017,CPU2_30018を装備しており、RAM10016には、図1の構成に加えて並列復号処理プログラムPROG5_30025が保存されている点異なる。

コンピュータA10002は、並列暗号化処理プログラムPROG4_30016を実行し、メッセージM10014の暗号文C10022を作成し、送信する。コンピュータB10003は、暗号文C10022を受信したあと、並列復号化処理プログラムPROG5_30025を実行し、もし改ざんが検出されなければ復号化結果をRAM10016に保存する。

並列暗号化処理プログラムPROG4_30016は、コンピュータA10002において、RAM10005から読み出されて、CPU1_30004とCPU2_30005により実行されることで実現される。並列暗号化処理プログラムPROG4_30016は、サブルーチンとして暗号化処理プログラムPROG1_10009,乱数生成処理PROG2_10010を内部で呼び出し、実行する。

並列復号化処理プログラムPROG5_30025は、コンピュータB10003において、RAM10016から読み出されてCPU1_30017とCPU2_30018により実行される。並列復号化処理プログラムPROG5_30025は、サブルーチンとして復号化処理プログラムPROG3_10020,乱数生成処理PROG2_10021を内部で呼び出し、実行する。

その他の構成、動作は図1と同じである。

【 0 0 5 4 】

並列暗号化処理プログラムPROG4_30016の処理の流れを図25を用いて説明する。記号「A||B」はビット列AとBの連結を示す。

ステップ40002:CPU1の実行するメッセージ処理によりメッセージMを M_1, M_2 に2分割する。

ステップ40003:CPU1が実行するPROG1_10009による暗号化処理により,初期値V+1,冗長性R+1,秘密鍵K,平文 M_1 ,を使って,暗号文 C_1 を出力する。

ステップ40004:CPU2が実行するPROG1_10009による暗号化処理により,初期値V+2,冗長性R+2,秘密鍵K,平文 M_2 ,を使って,暗号文 C_2 を出力する。

ステップ40005:CPU1が実行するPROG1_10009による暗号化処理により,初期値V,冗長性R,秘密鍵K,平文($R_1 || R_2$),を使って,暗号文 C_3 を出力する。

ステップ40006:暗号文Cの生成。 $C = C_1 || C_2 || C_3$ とする。

ステップ40007:暗号文Cをメモリへ保存

並列復号化処理プログラムPROG5_30025の処理の流れを図26を用いて説明する。

ステップ40102:CPU1の実行する暗号文処理により暗号文C'を C'_1, C'_2, C'_3 に3分割する。 C'_3 は192ビットであり, C'_1, C'_2 は同じ長さ,かつ $C' = C'_1 || C'_2 || C'_3$ である。

ステップ40103:CPU1が実行するPROG3_10020による復号化処理により,初期値V+1,冗長性R+1,秘密鍵K,暗号文ブロック C'_1 ,を使って,復号化を行い結果を保存する。

ステップ40104:CPU2が実行するPROG3_10020による復号化処理により,初期値V+2,冗長性R+2,秘密鍵K,暗号文ブロック C'_2 を使って,復号化を行い,結果を保存する。

ステップ40105:ステップ40103の復号化結果,ステップ40104の復号化結果のうち少なくともひとつが非受理ならばステップ40111を実行する。

ステップ40106: CPU1が実行するPROG3_10020による復号化処理により,初期値V,冗長性R,秘密鍵K,暗号文ブロック C'_3 を使って,復号化を行い,結果を保存する。

ステップ40107:ステップ40106の復号化結果が非受理ならばステップ40111を実行する。

ステップ40108:ステップ40106の復号化結果が(R+1) || (R+2)でないなら,ステップ40111を実行する。

ステップ40109:復号化結果 $M' = M'_1 || M'_2$ とする。

ステップ40110:M' をメモリへ格納し、ステップ40112を実行する。

ステップ40111:非受理を出力する。

【 0 0 5 5 】

実施例4により独立な二つのプロセッサを使った並列暗号処理が可能となる。

図27は、上述の並列化手法で用いた暗号化処理の説明図である。

メッセージM401402を分割して生成した M_1 -40141, M_2 -40142にそれぞれ冗長性, $R+1$, $R+2$ を付加した,40143,40144を暗号化処理40146,40147により暗号化,暗号文ブロック C_1 -40149, C_2 -40150を得る。

さらに冗長性 $R+1$, $R+2$ をメッセージとして冗長性を R 使って暗号化し,暗号文ブロック C_3 -40151を得る。

暗号文ブロック C_1 -40149, C_2 -40150, C_3 -40151を連結し暗号文C40152として出力する。

【 0 0 5 6 】

図28は、対応する並列復号化処理の説明図である。

暗号文C' 40160を3分割し, C'_1 -40161, C'_2 -40162, C'_3 -40163を生成する。

それぞれを復号化処理40164,40165,40166により復号化,平文ブロック40167,40168,40169を得る。

これらすべてが受理であって,平文ブロック40167,40168の冗長性が,平文ブロック40169のメッセージ部と同じであり,さらに平文ブロック40169が予め共有されている冗長性であれば,平文ブロック40167,40168からそれぞれメッセージ部 M'_1 -40170, M'_2 -40171を取り出し,これらを連結してメッセージM' 40172を得る。

【 0 0 5 7 】

以上の各実施例におけるCPUはプログラムを実行するものであれば、汎用、専用を問わず用いることができる。また、各実施例は、CPUがプログラムを実行することにより実現される例であったが、各処理を専用のハードウェアを用いても、高速にかつ、低価格で実現することも可能である。

【 0 0 5 8 】

【発明の効果】

より安全で高速な共通鍵暗号技術を提供することが可能となる。

【図面の簡単な説明】

【図1】

各実施例のシステム構成図を示す。

【図2】

平文準備サブルーチンのフロー図を示す。

【図3】

乱数生成サブルーチンのフロー図を示す。

【図4】

暗号化サブルーチンのフロー図を示す。

【図5】

図1の復号化処理プログラムのフロー図を示す。

【図6】

図7の暗号文準備サブルーチンのフロー図を示す。

【図7】

図7の復号化サブルーチンのフロー図を示す。

【図8】

図7の平文切出しサブルーチンのフロー図を示す。

【図9】

図7の冗長性切出しサブルーチンのフロー図を示す。

【図10】

暗号化処理の、データブロックによる図を示す。

【図11】

図7で示した復号化処理の、データブロックによる図を示す。

【図12】

実施例2の乱数生成2サブルーチンのフロー図を示す。

【図13】

実施例2の暗号化2サブルーチンのフロー図を示す。

【図14】

実施例2の復号化処理プログラムのフロー図を示す。

【図 1 5】

実施例2の復号化2サブルーチンのフロー図を示す。

【図 1 6】

実施例2の暗号化処理の、データブロックによる図を示す。

【図 1 7】

実施例2の復号化処理の、データブロックによる図を示す。

【図 1 8】

実施例3の暗号化処理プログラムのフロー図を示す。

【図 1 9】

実施例3の乱数生成3サブルーチンのフロー図を示す。

【図 2 0】

実施例3の暗号化3サブルーチンのフロー図を示す。

【図 2 1】

実施例3の復号化処理プログラムのフロー図を示す。

【図 2 2】

実施例3の復号化3サブルーチンのフロー図を示す。

【図 2 3】

実施例3の暗号化処理の、データブロックによる図を示す。

【図 2 4】

実施例3の復号化処理の、データブロックによる図を示す。

【図 2 5】

実施例4の並列暗号化処理プログラムのフロー図を示す。

【図 2 6】

実施例4の並列復号化処理プログラムのフロー図を示す。

【図 2 7】

実施例4の暗号化処理の、データブロックによる図を示す。

【図 2 8】

実施例4の復号化処理の、データブロックによる図を示す。

【符号の説明】

10001,30001:ネットワーク

10002,10003,30002,30003:コンピュータ

10004,10015,30004,40005,30017,30018:CPU

10005,10016,30006,30019:メモリ

10006,10017,30007,30020:ネットワークインターフェース

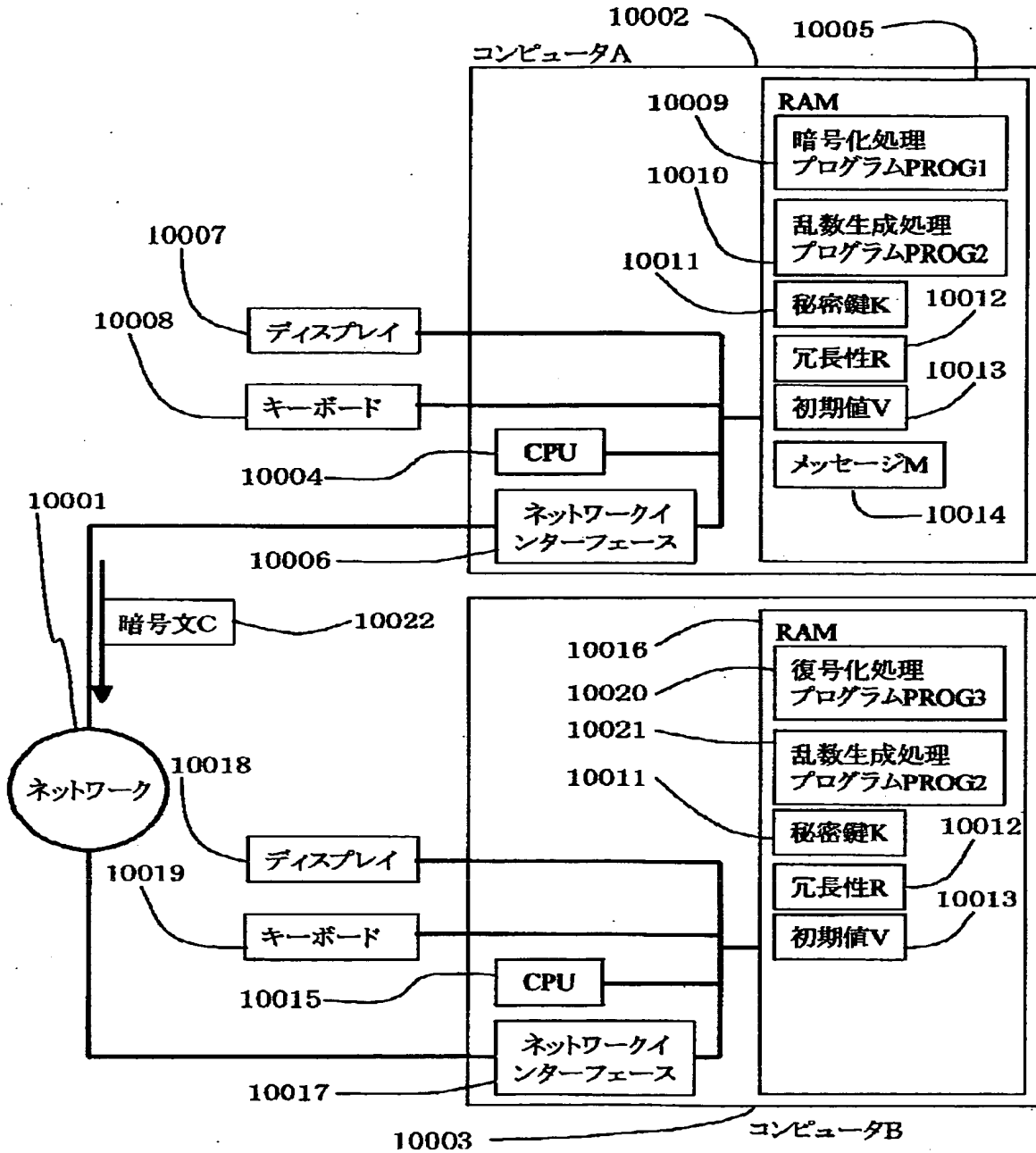
10007,10018,30008,30021:ディスプレイ

10008,10019,30009,30022:キーボード

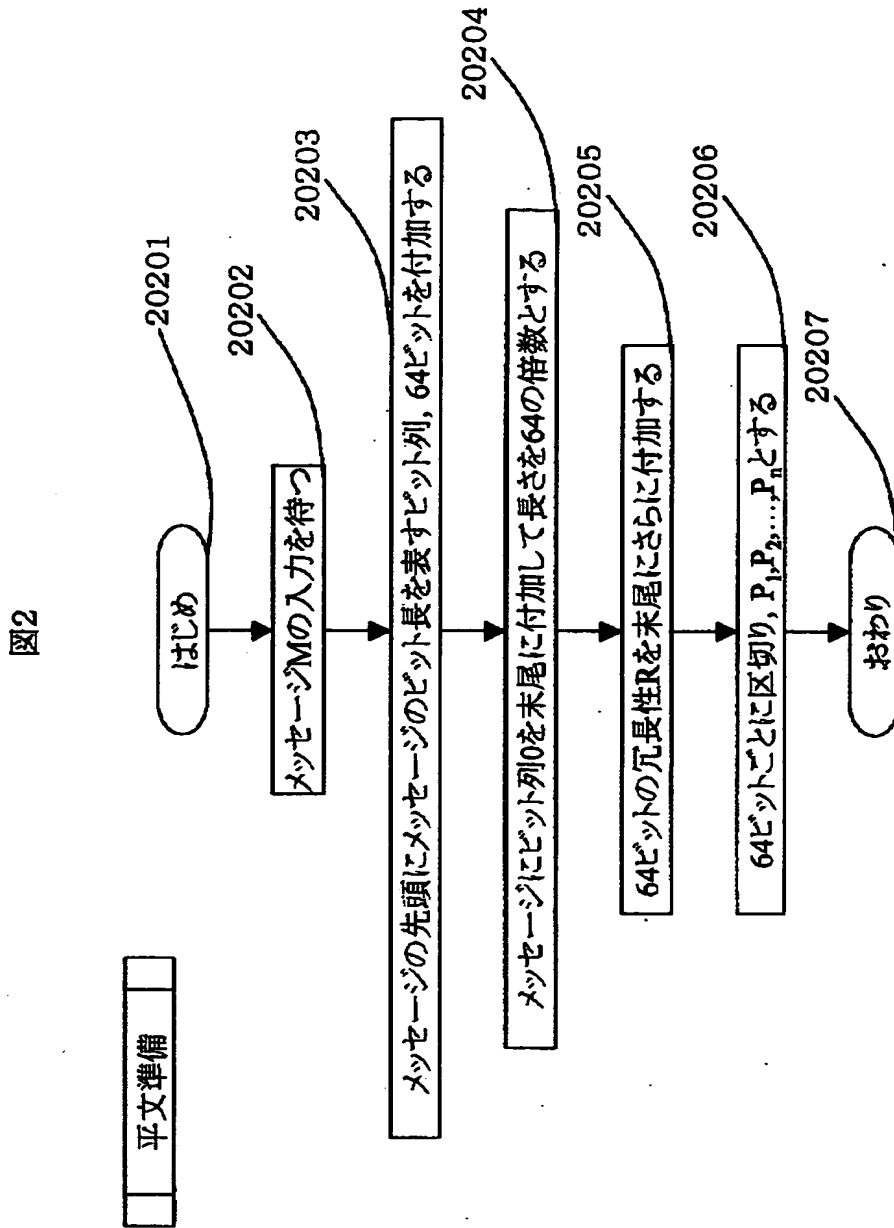
【書類名】 図面

【図1】

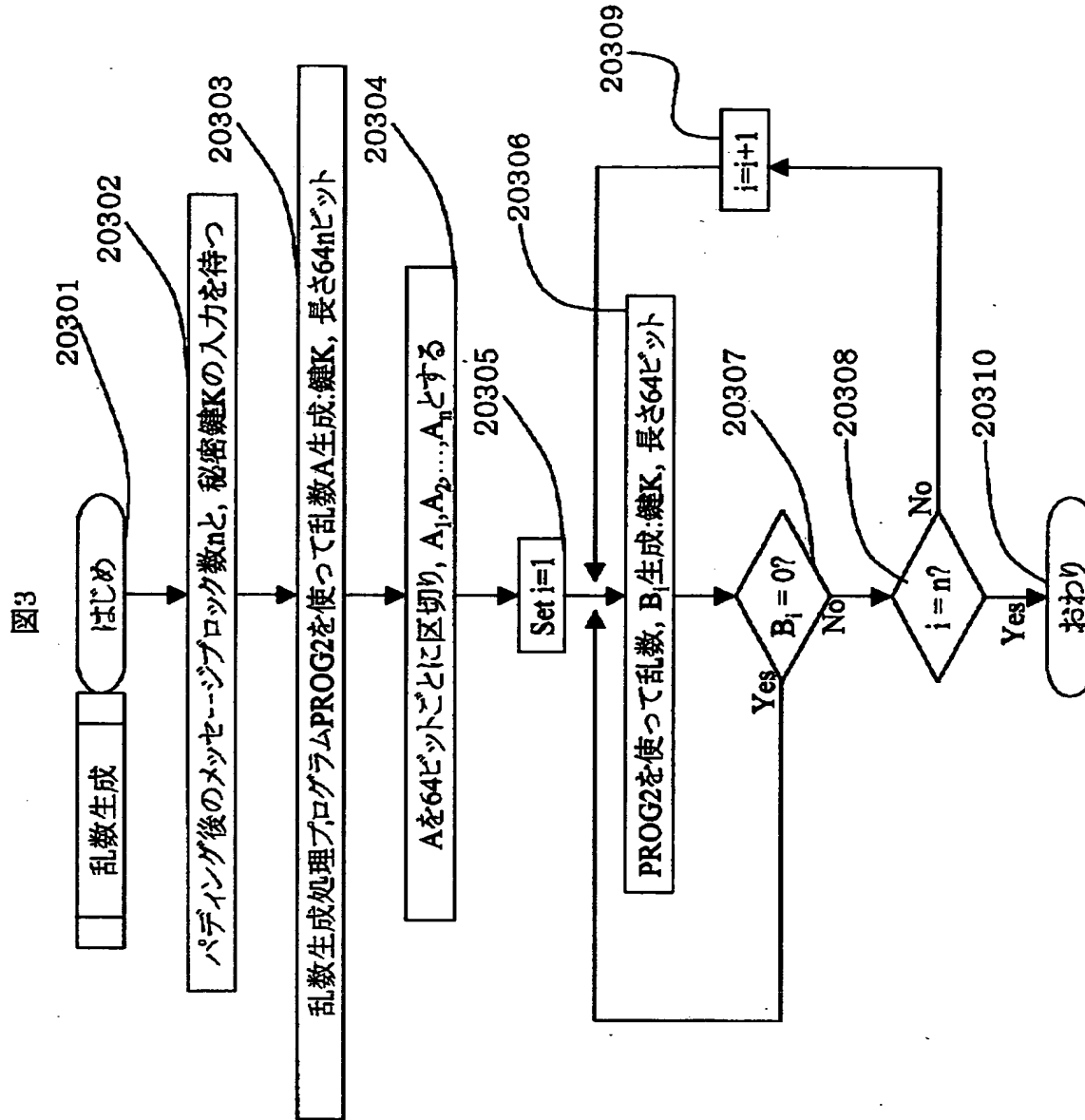
図1



【図2】

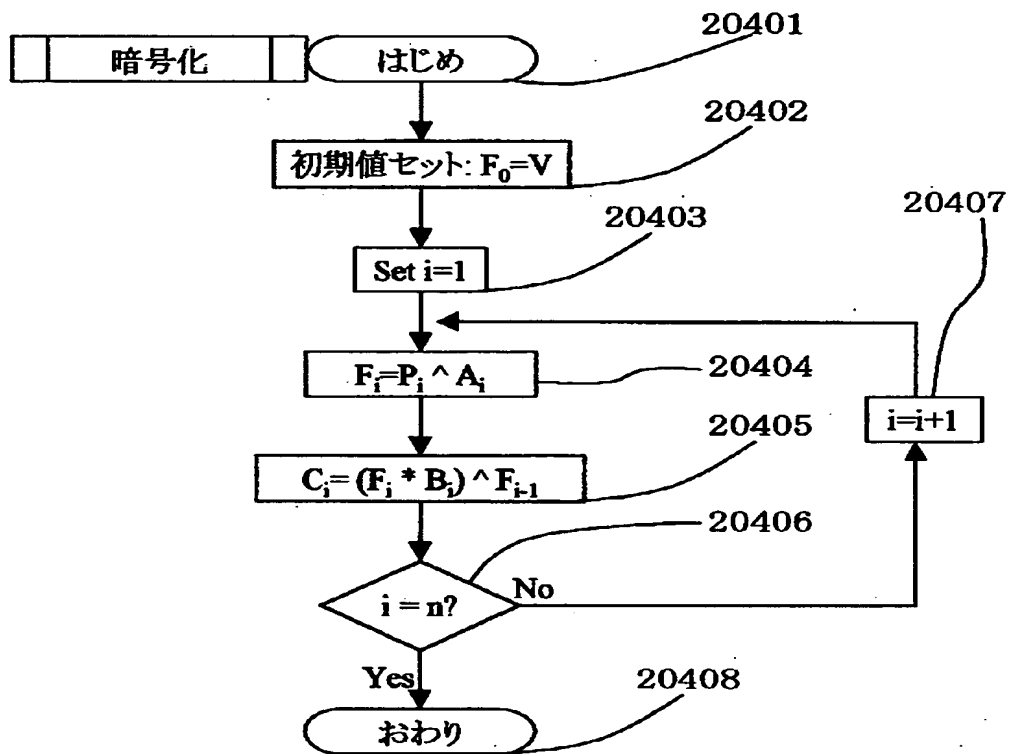


【図 3】



【図 4】

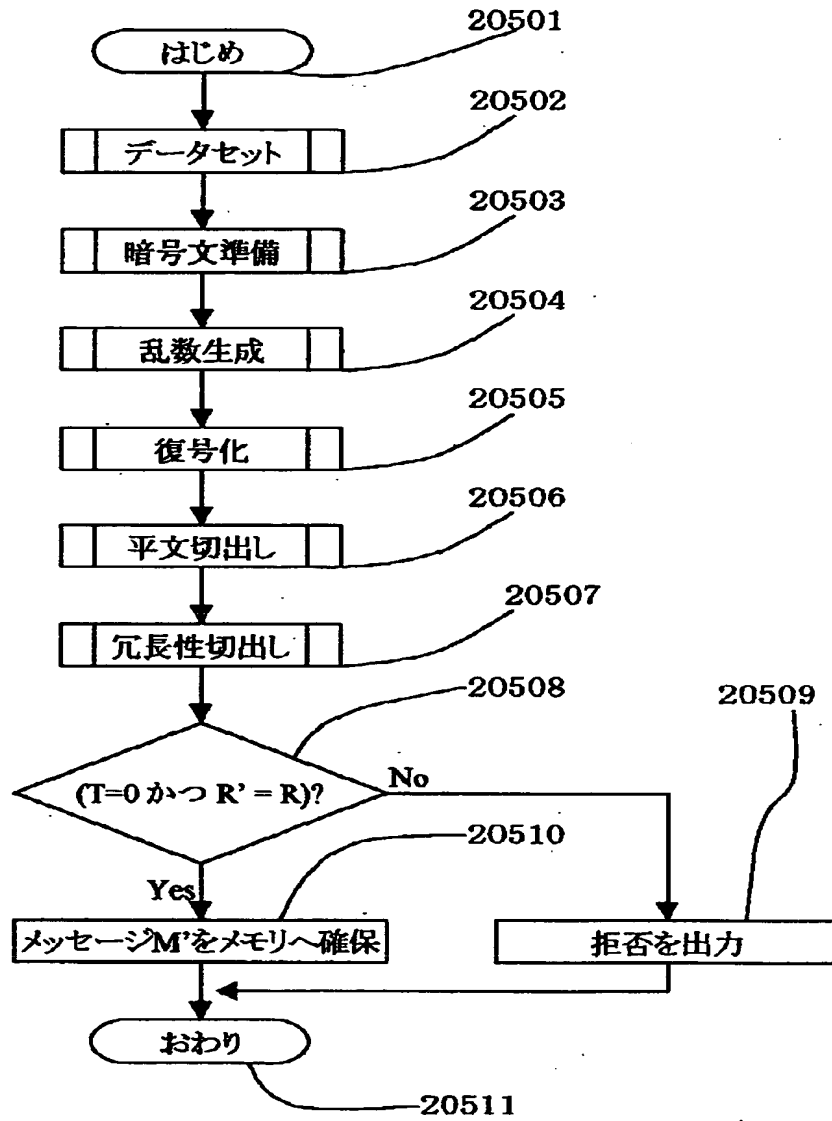
図4



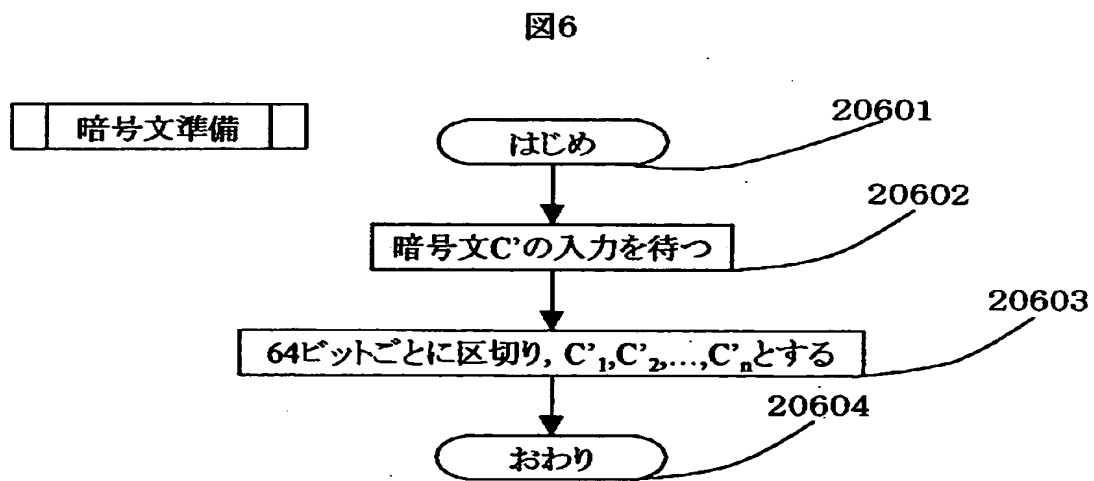
【図 5】

図5

復号化処理
プログラムPROG3

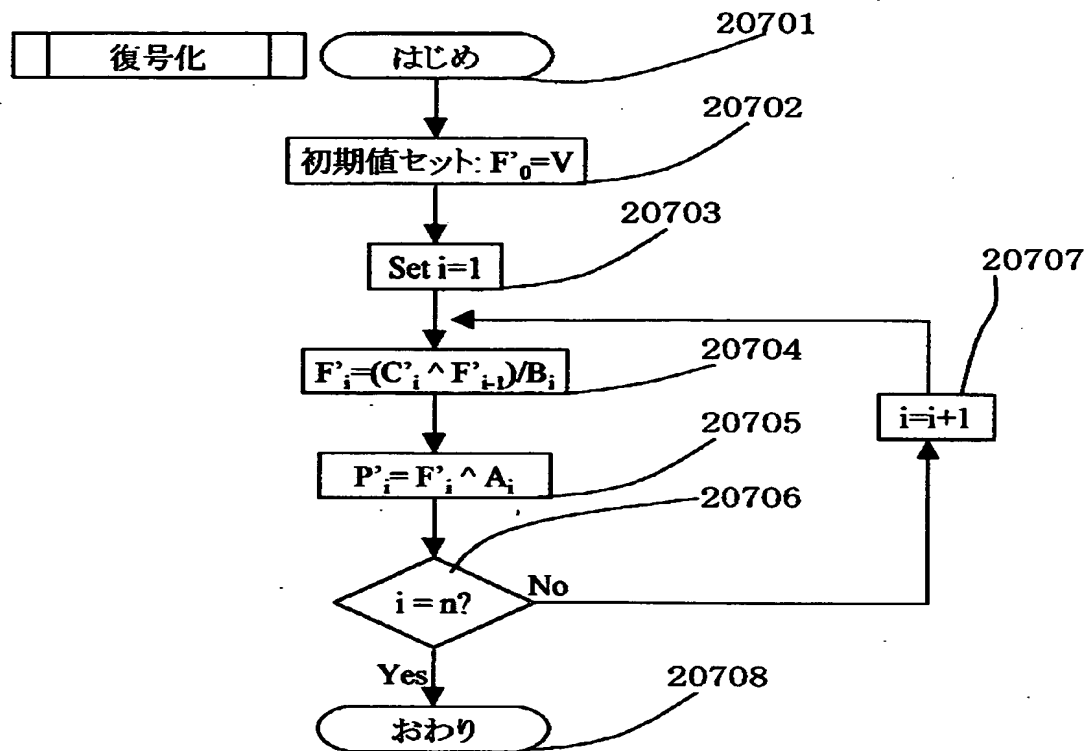


【図 6】



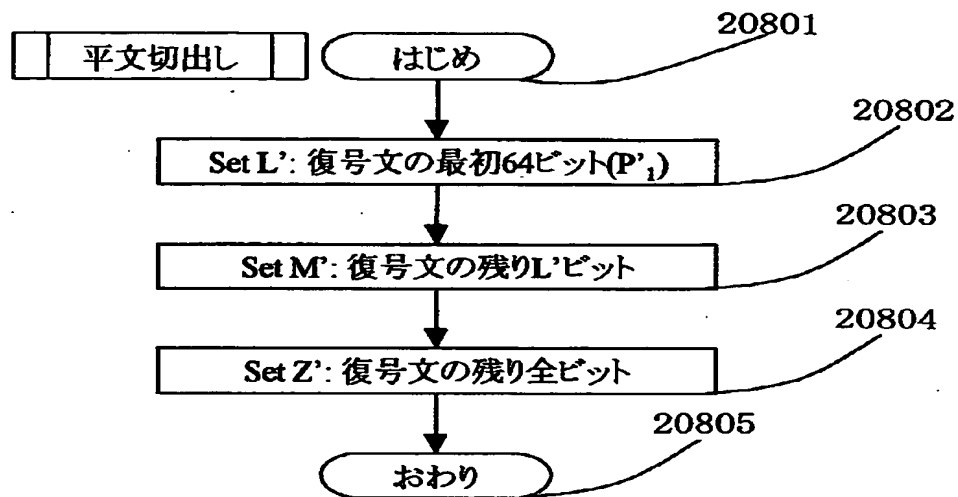
【図7】

図7



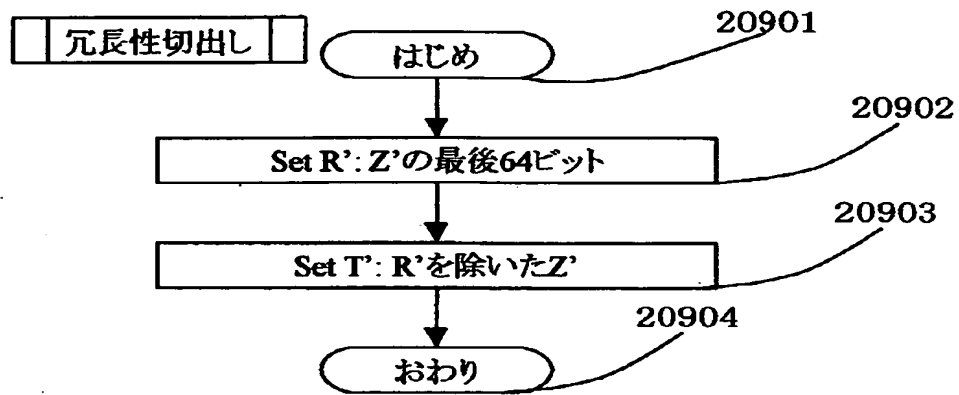
【図 8】

図8



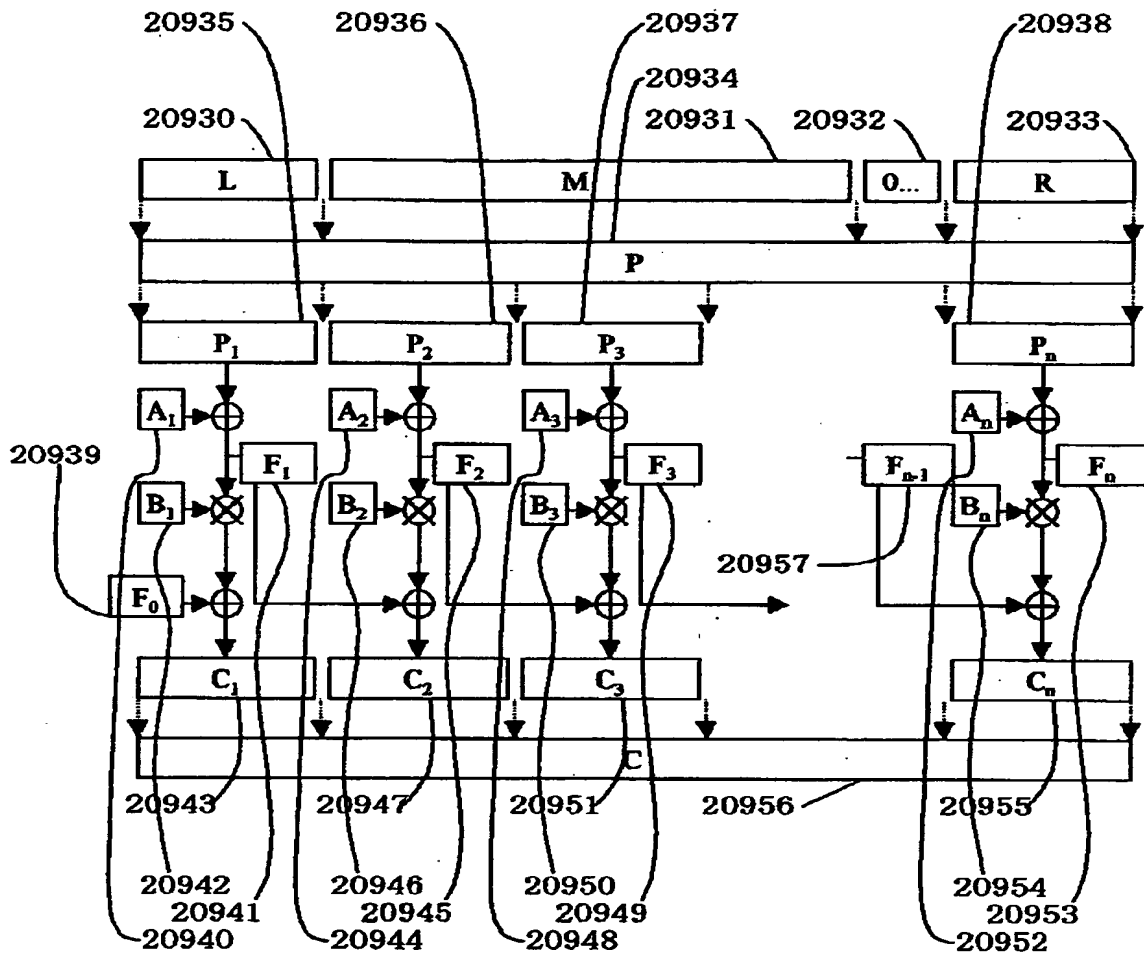
【図 9】

図9



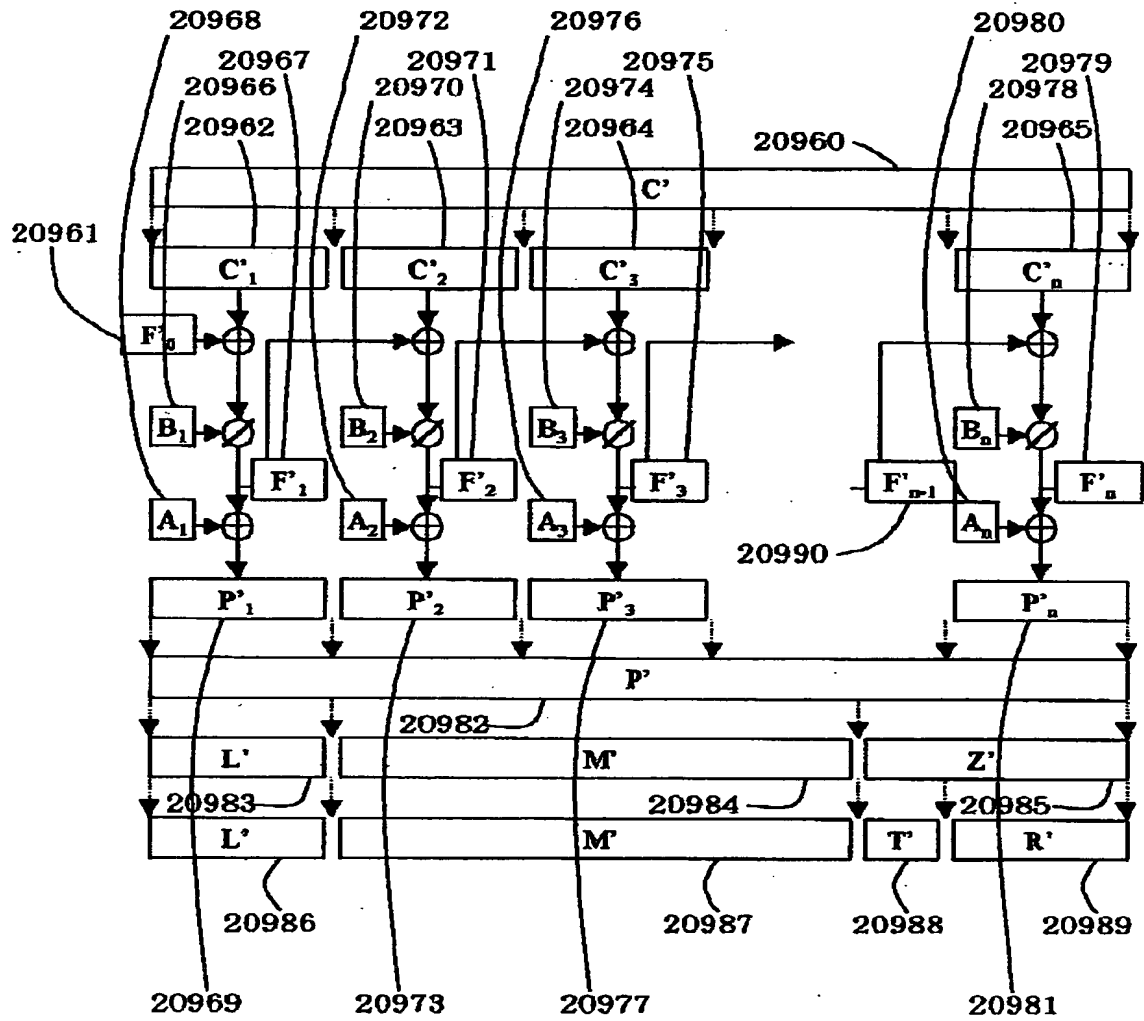
【図10】

図10



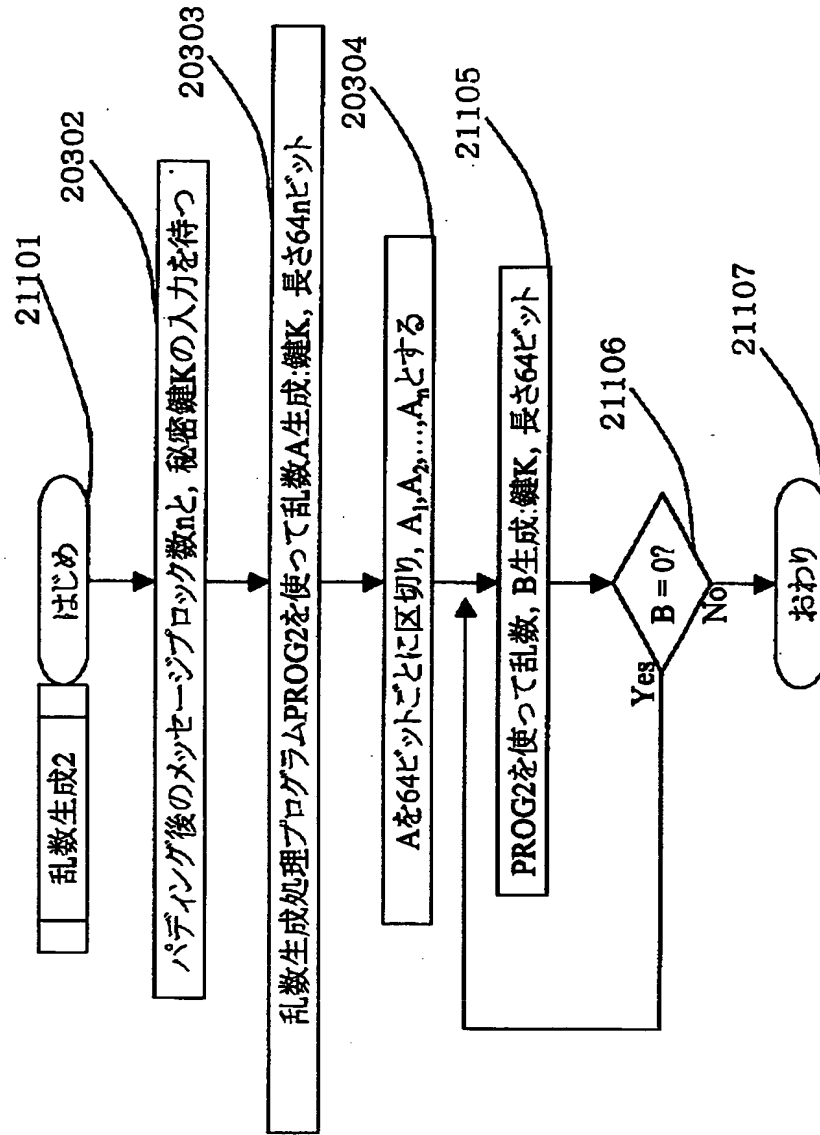
【図11】

図11



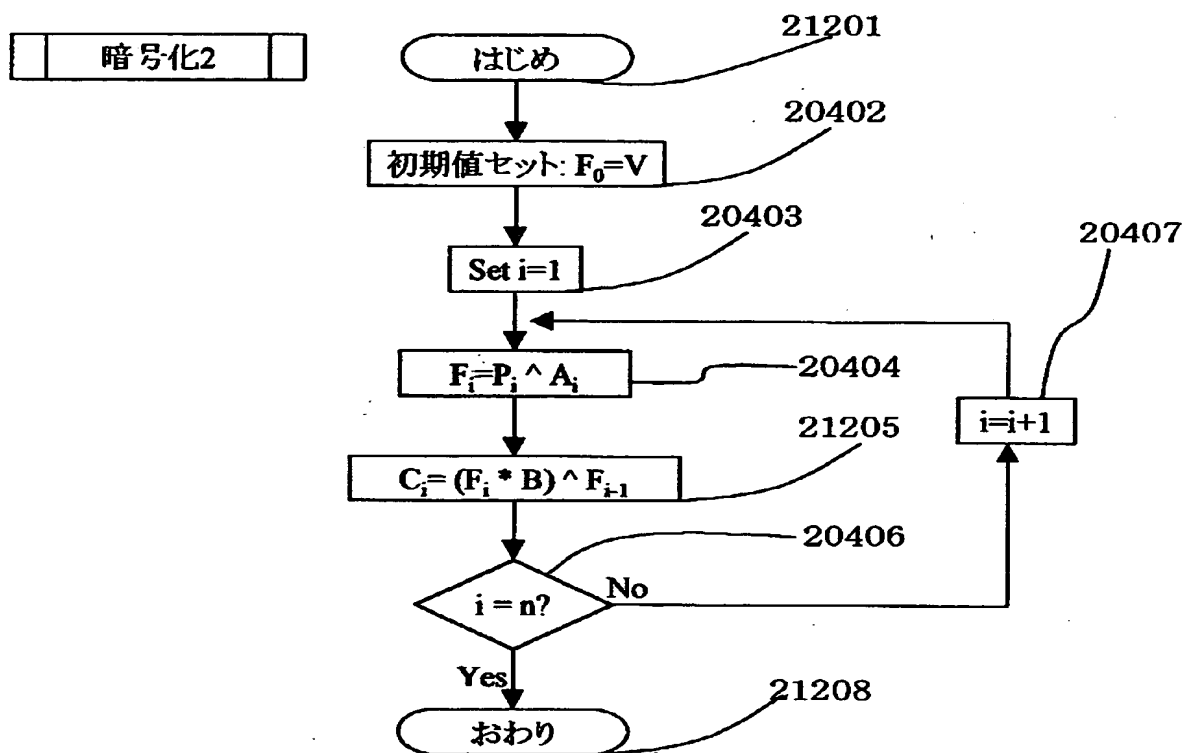
【図 12】

図12



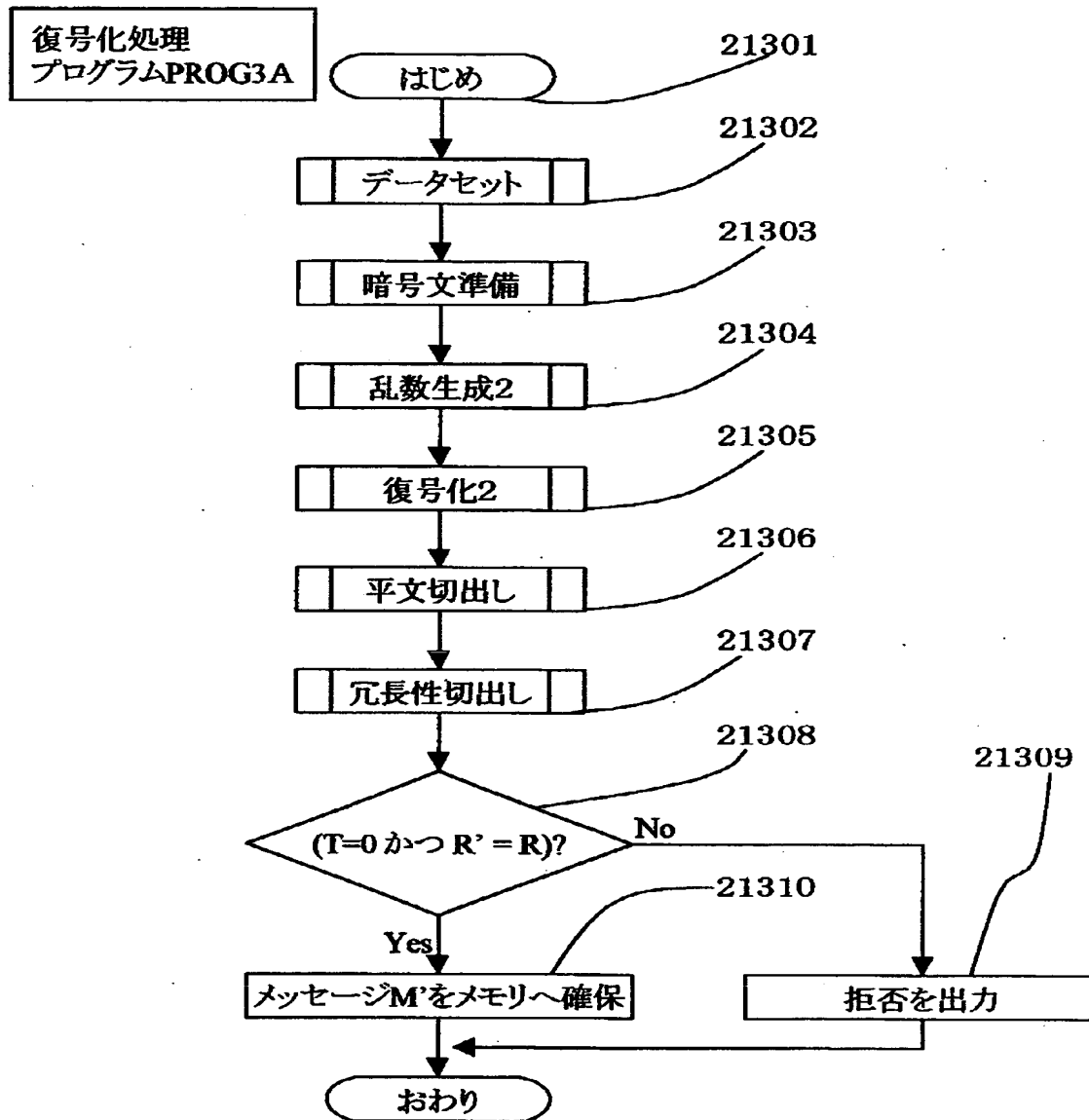
【図 1 3】

図13



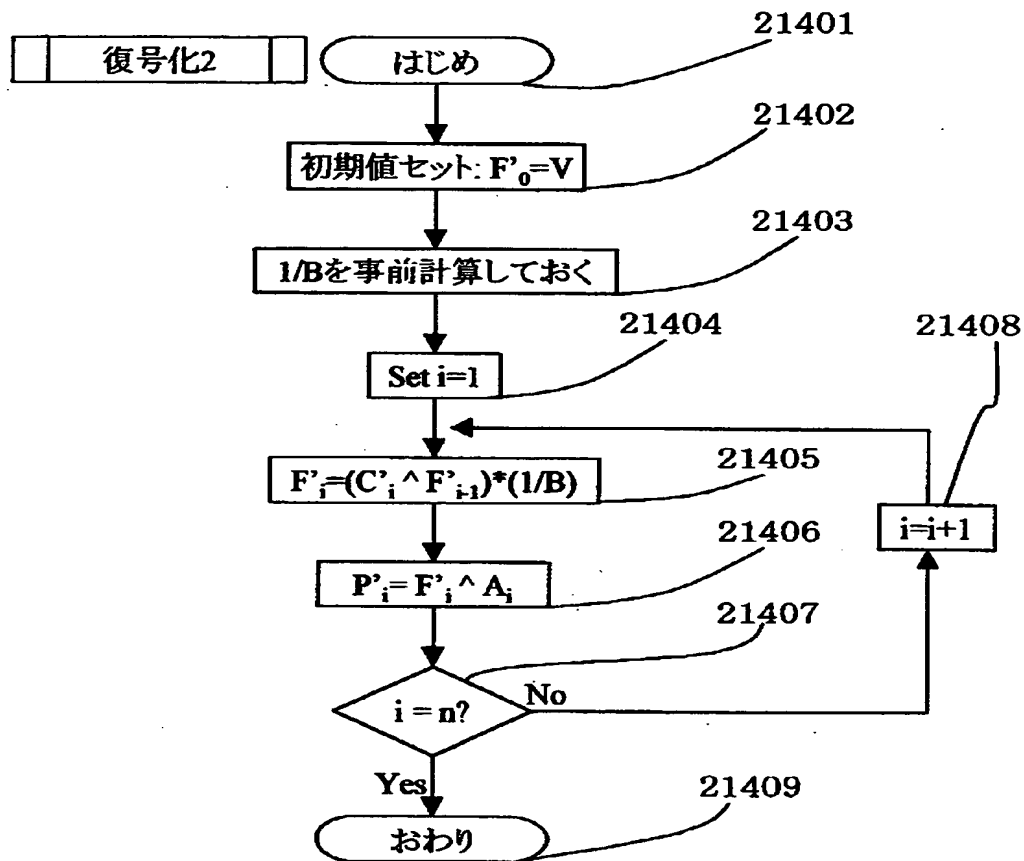
【図14】

図14



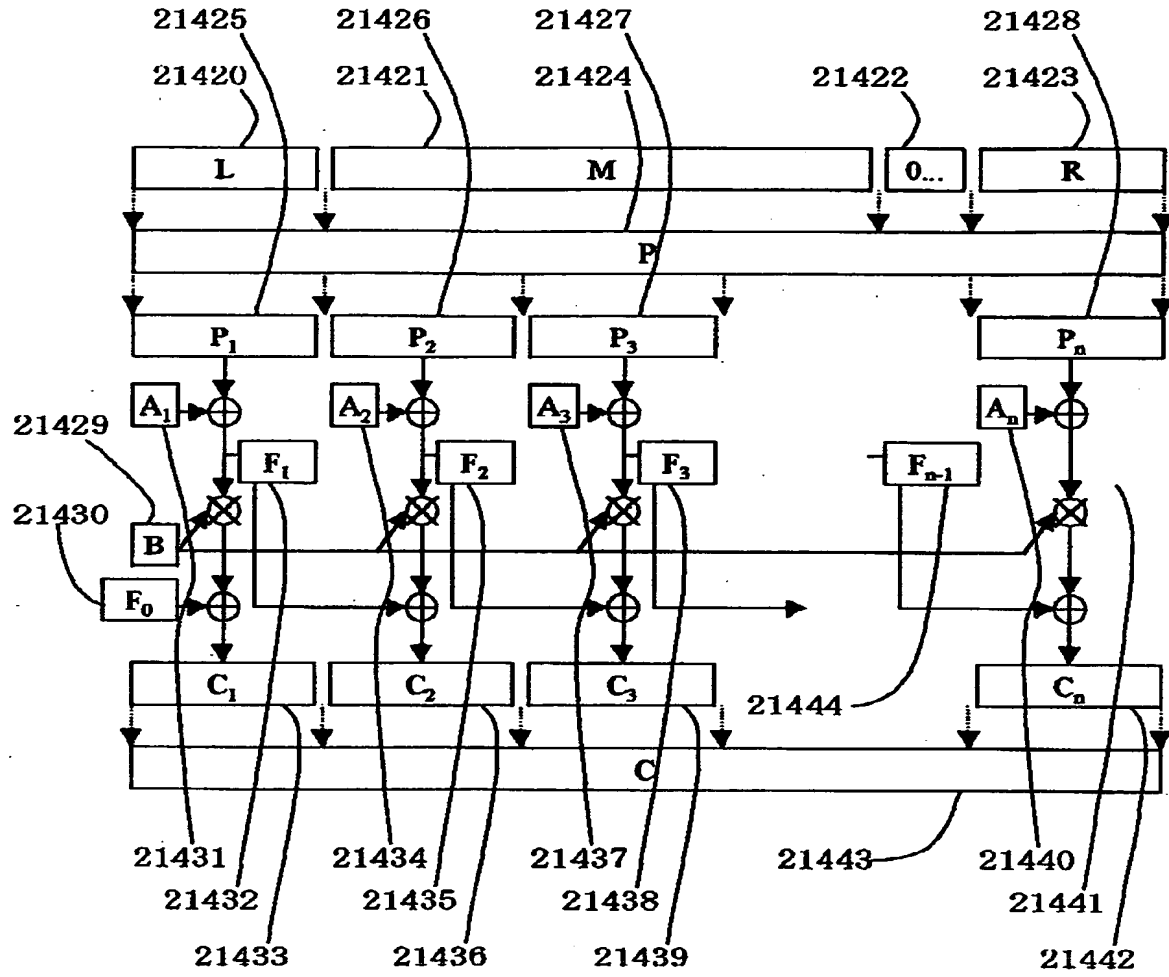
【図 15】

図15



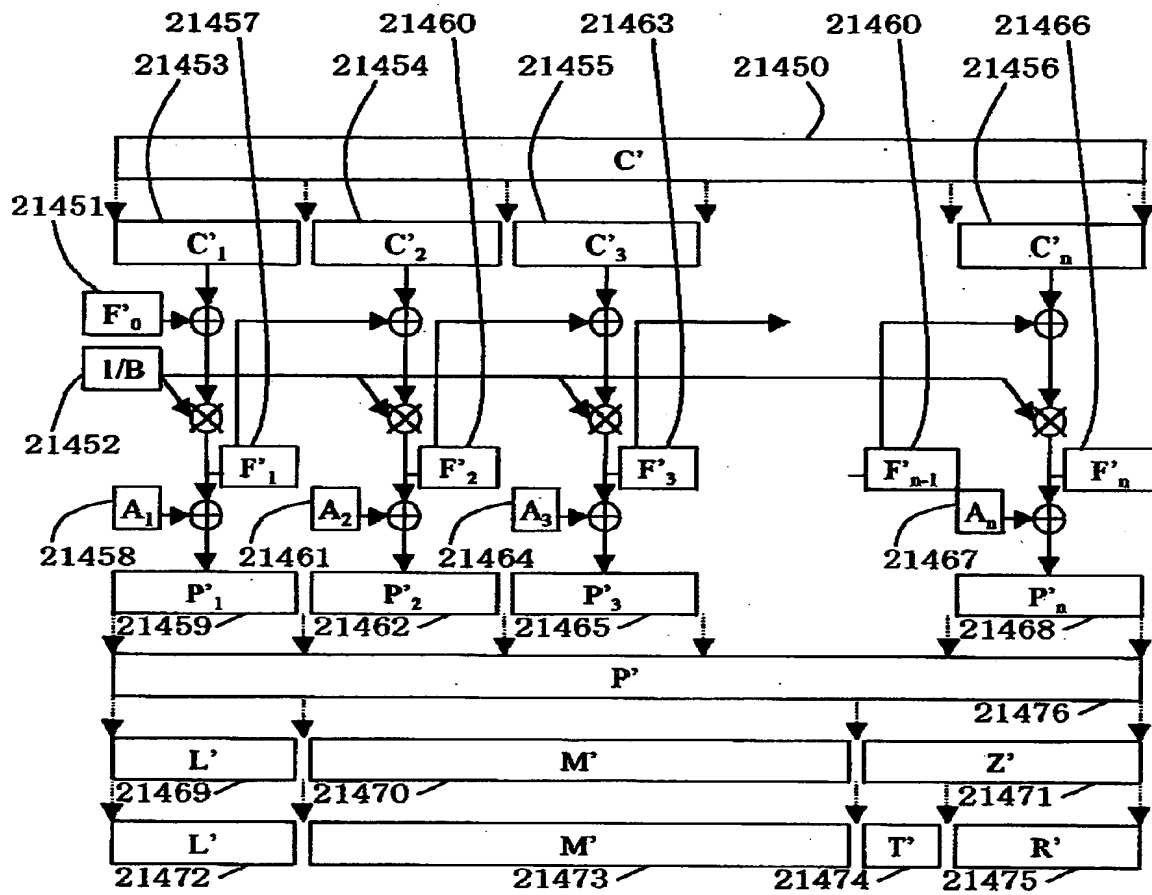
【図16】

図16



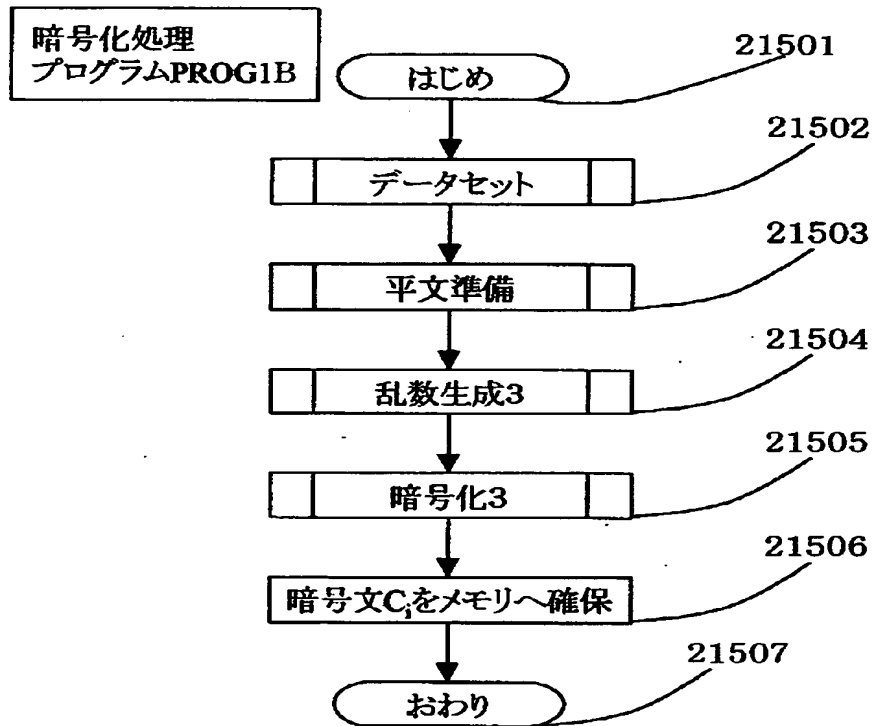
【図17】

図17



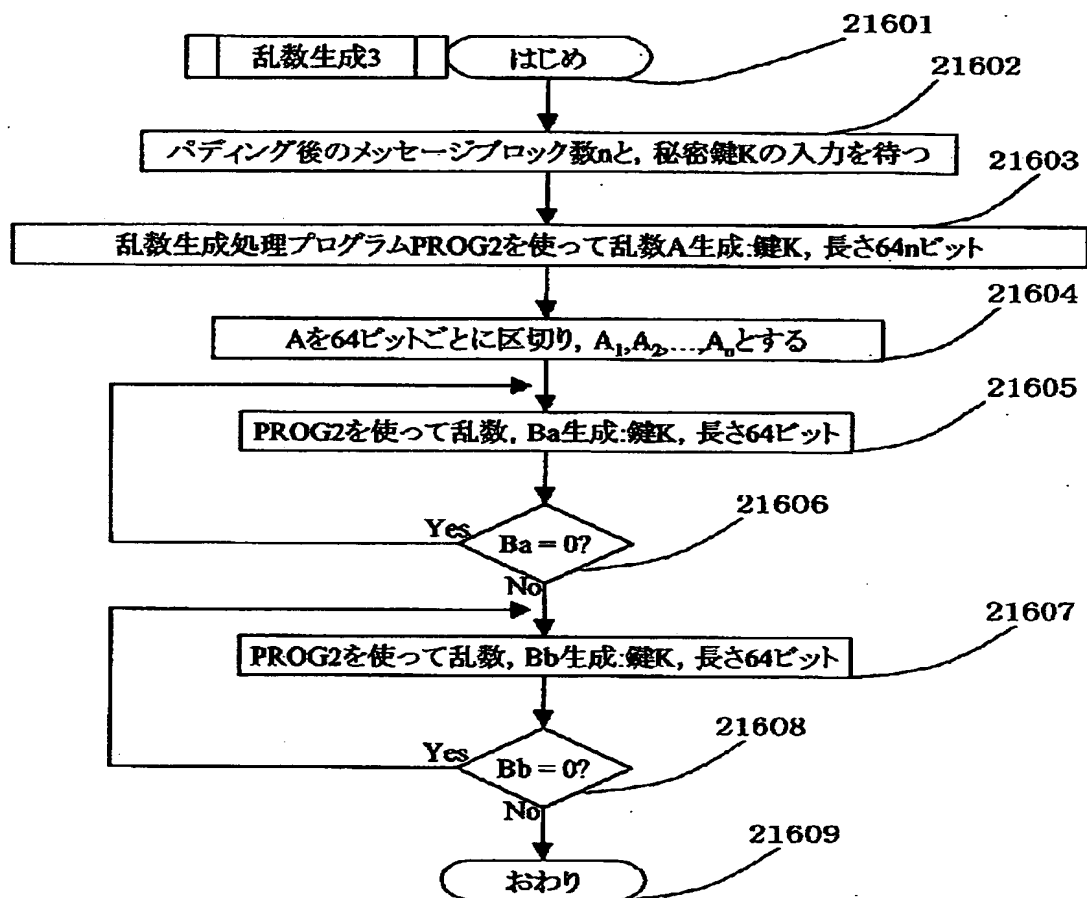
【図18】

図18



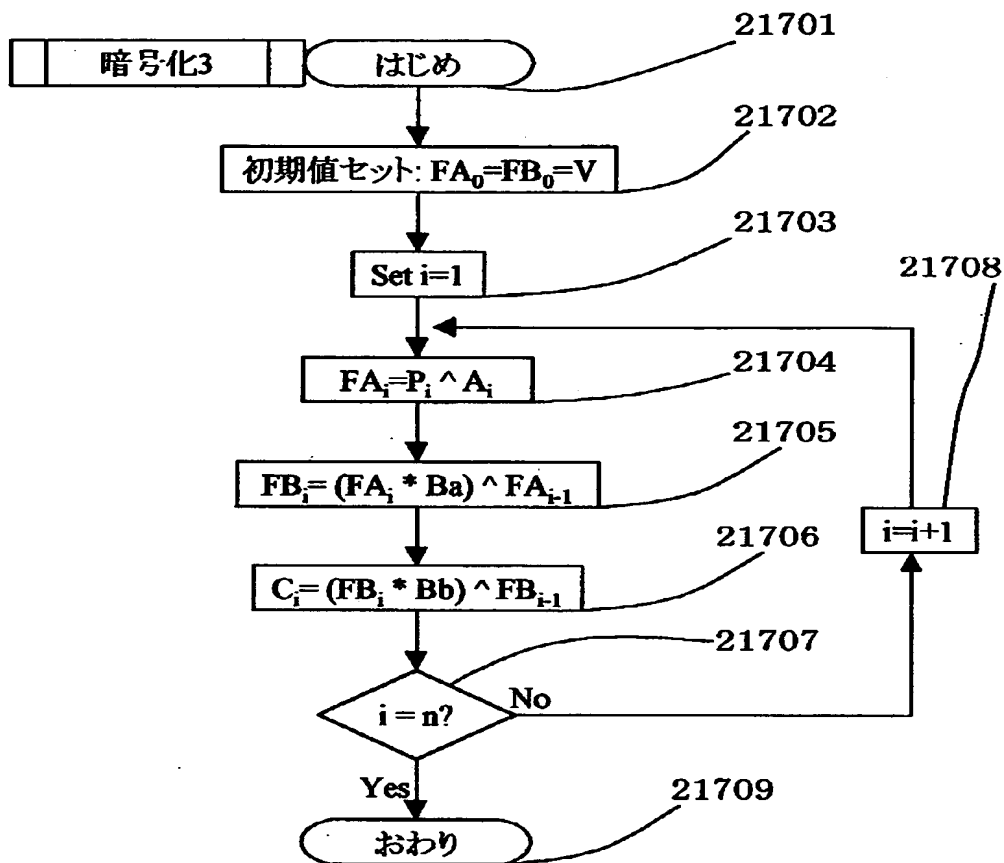
【図19】

図19



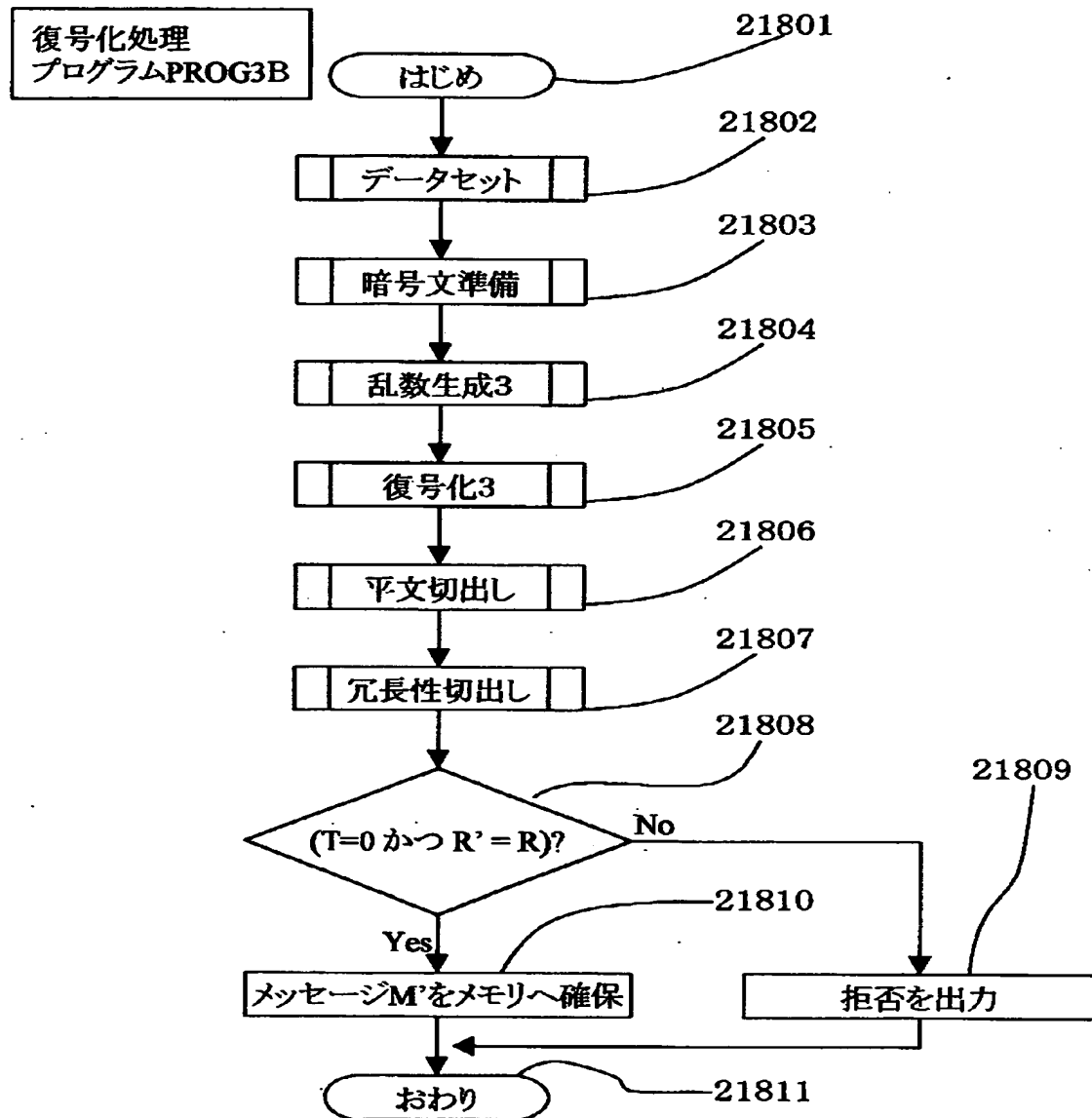
【図20】

図20



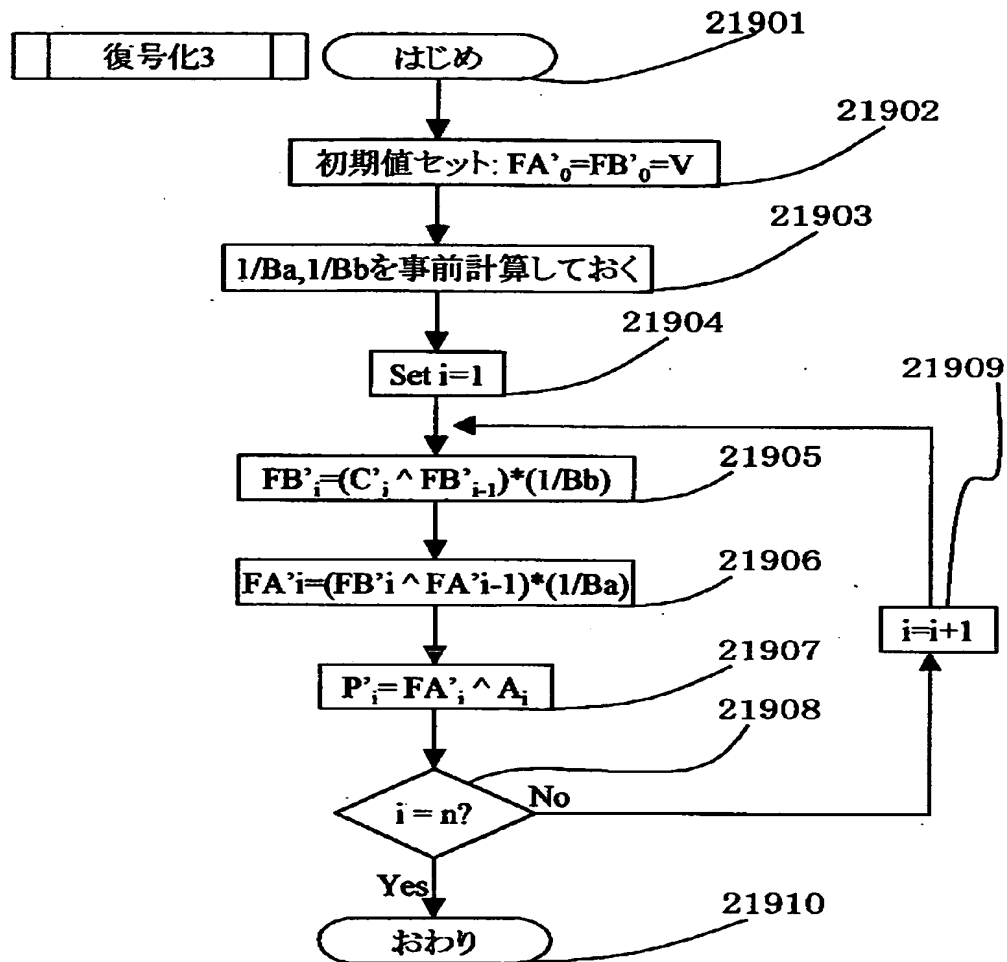
【図21】

図21



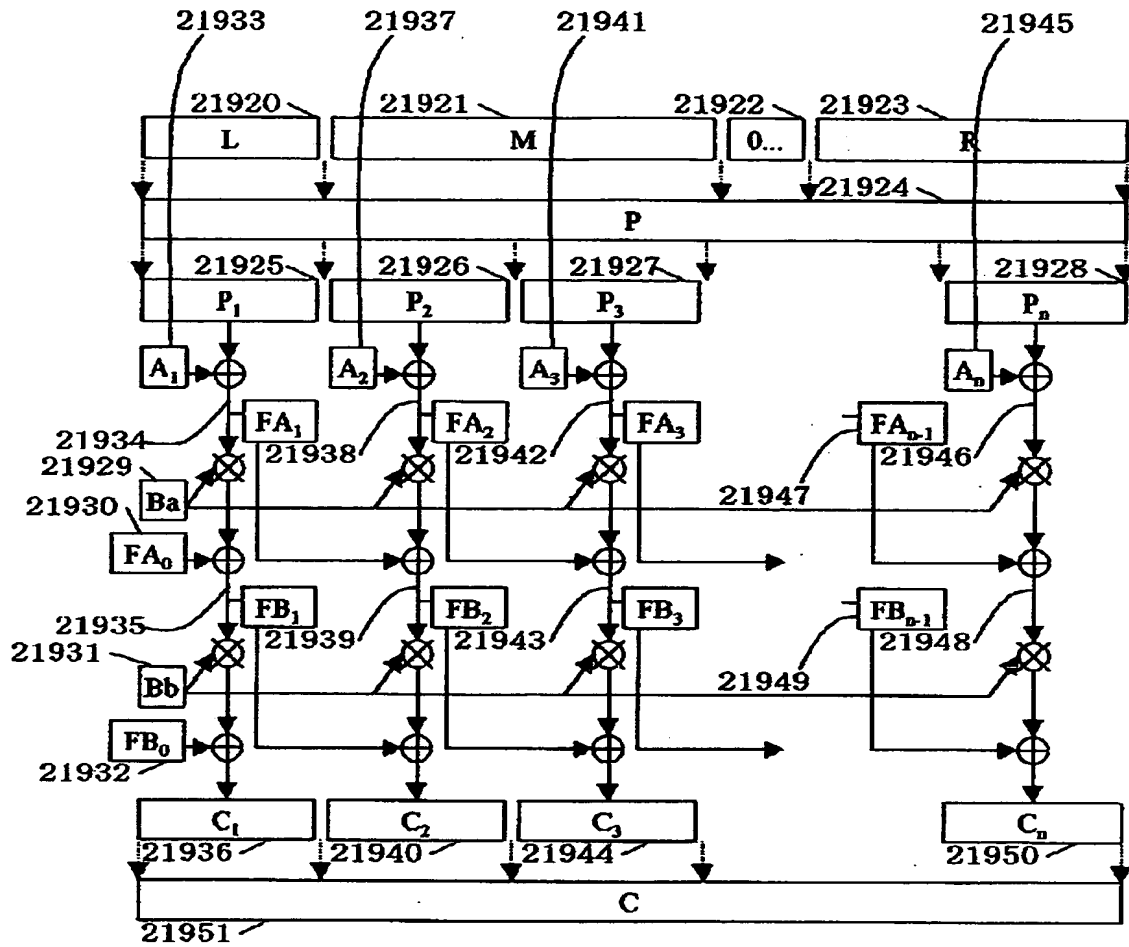
【図 22】

図22



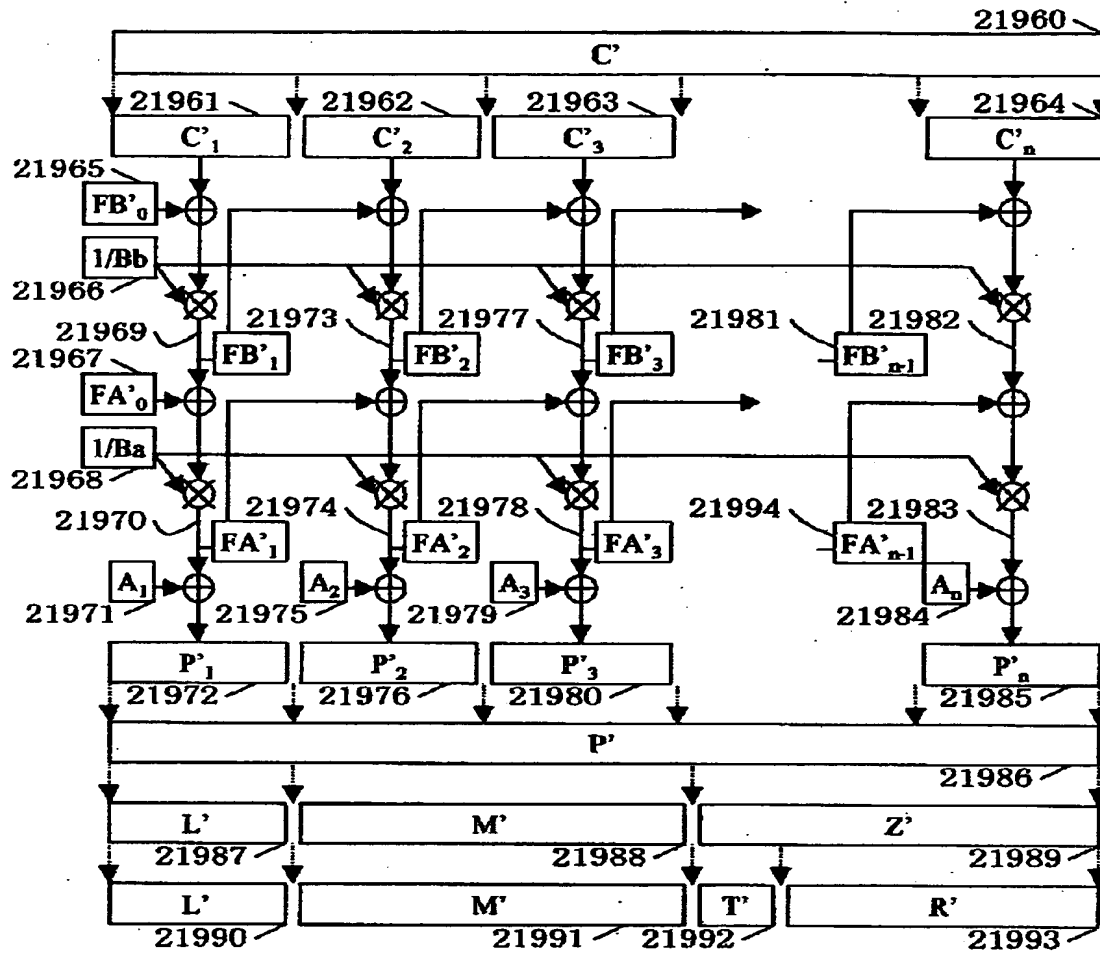
【図 23】

図23



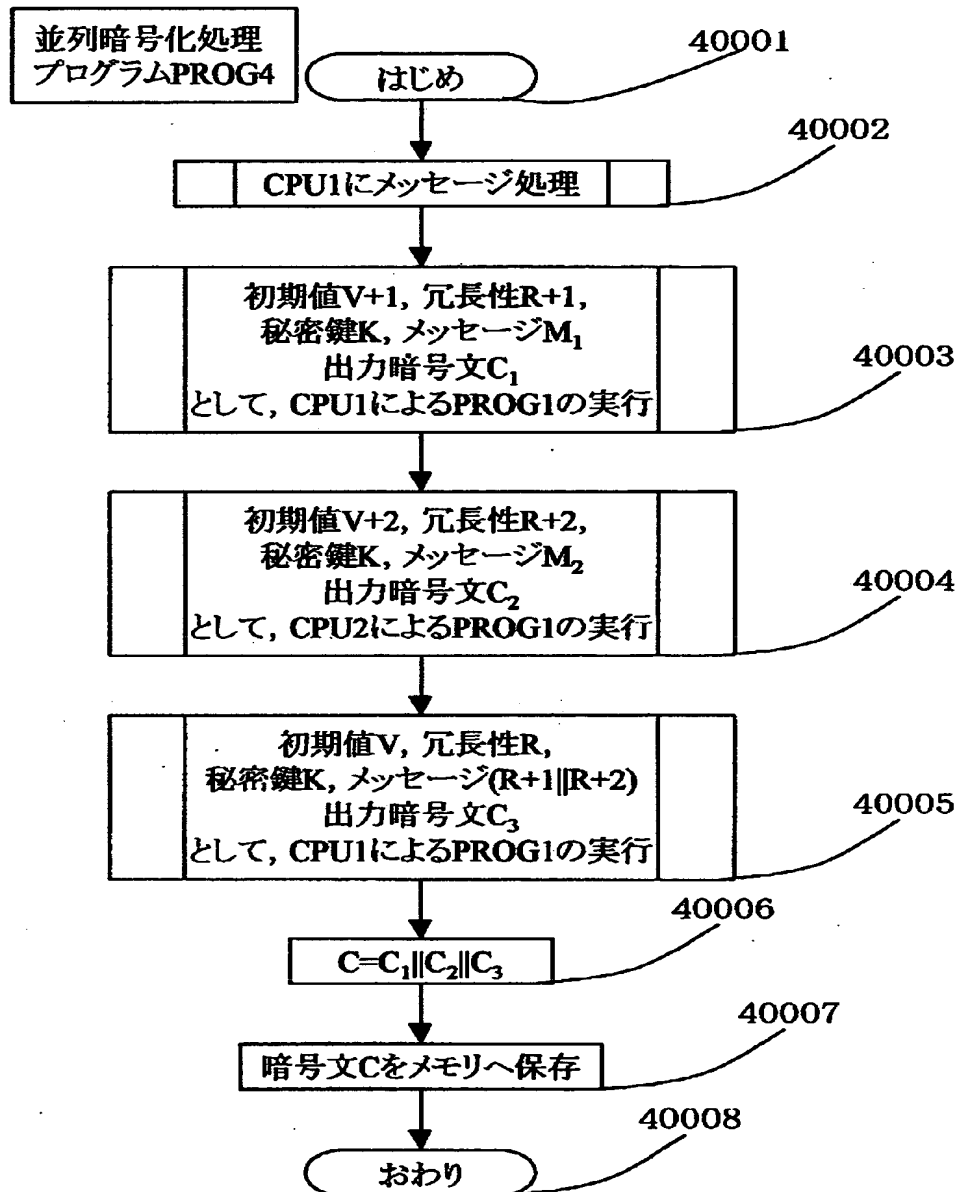
【図24】

図24



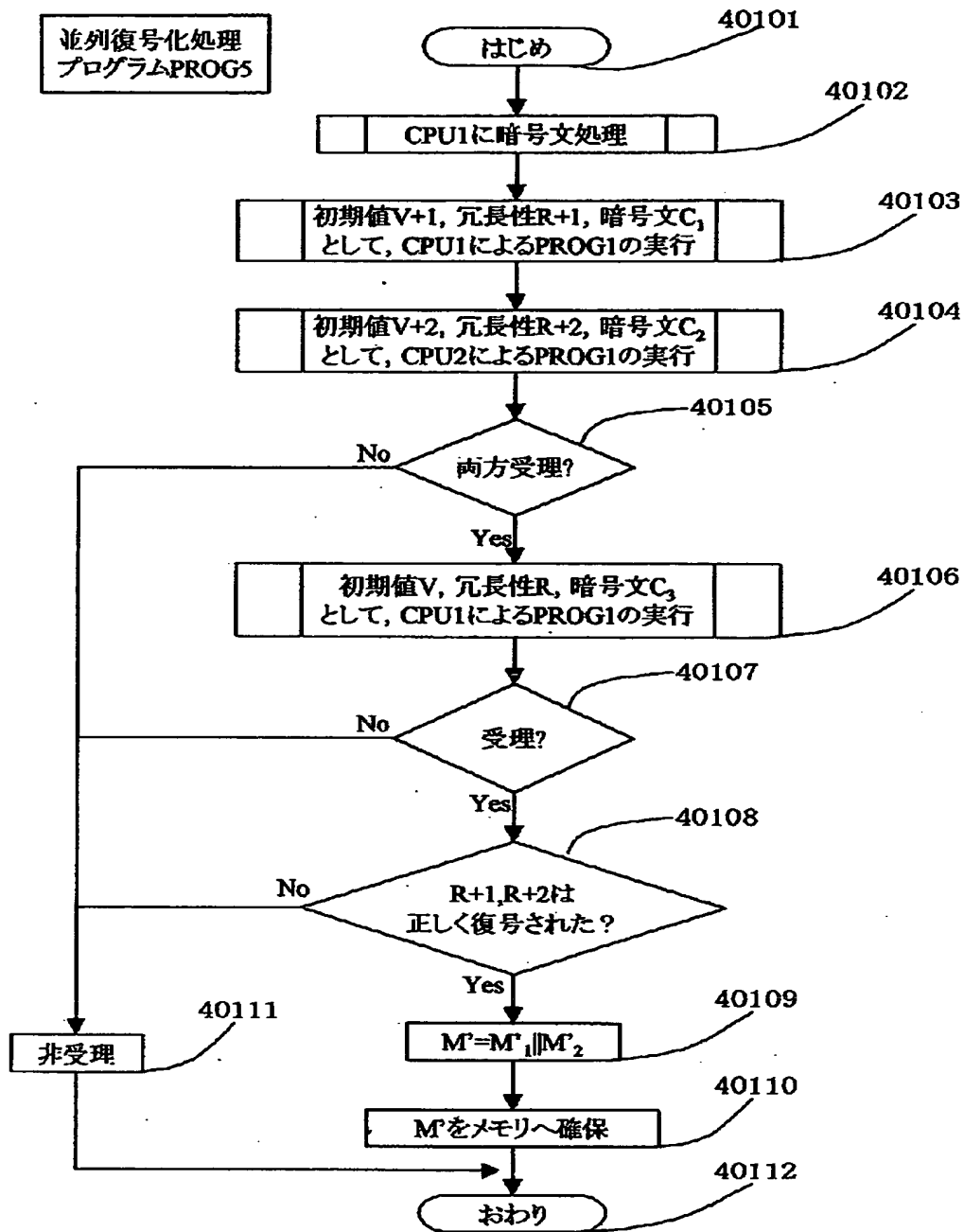
【図 25】

図25



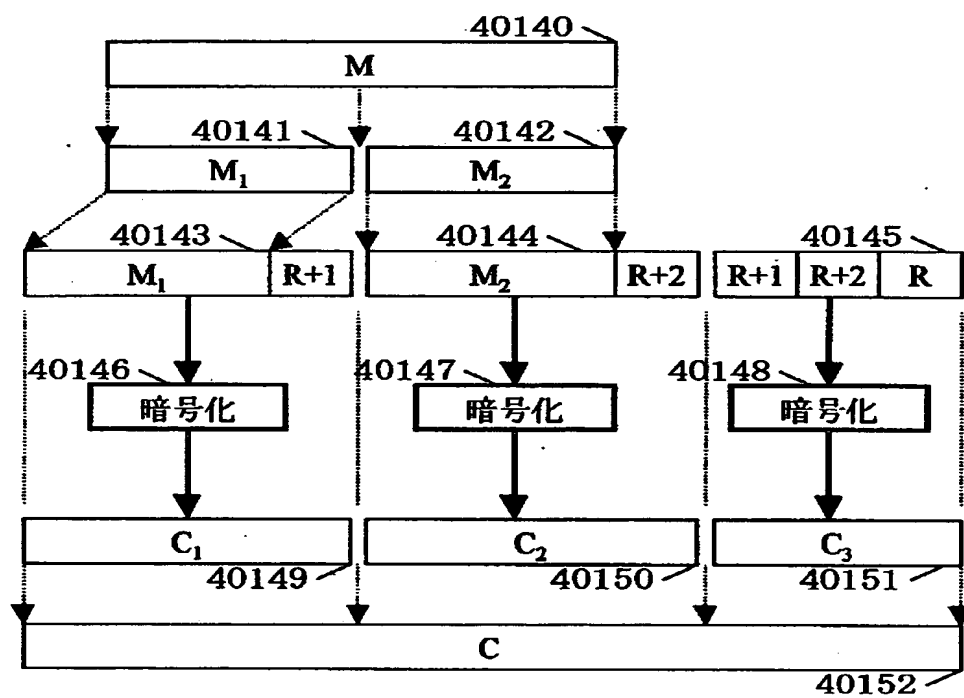
【図 26】

図26



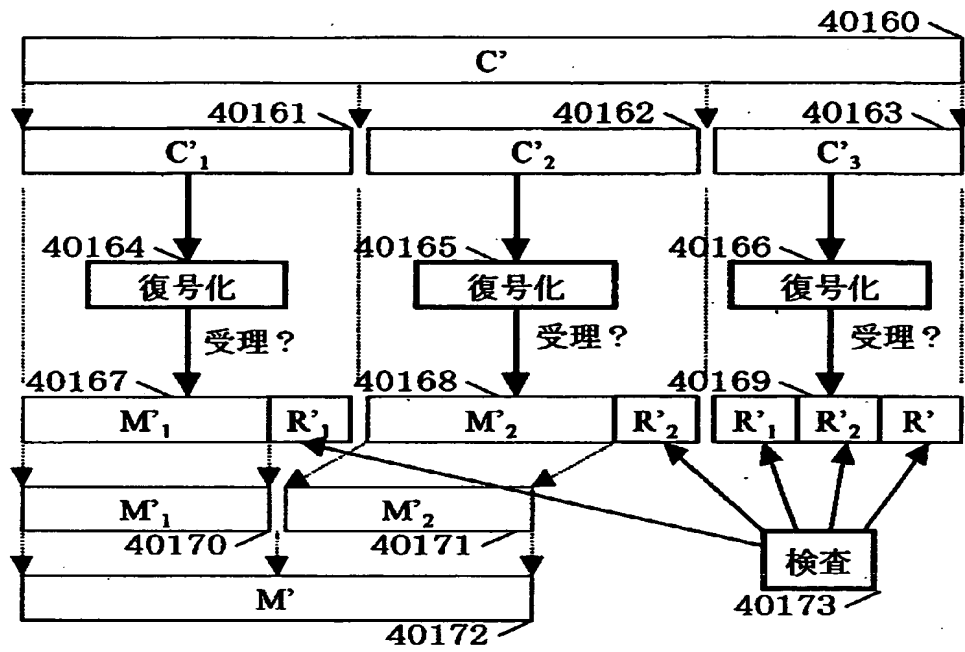
【図 2 7】

図27



【図 28】

図28



【書類名】 要約書

【要約】

【課題】

高速で、並列度の高い暗号処理と改ざん検出を実現できる共通鍵暗号技術を提案すること。

【解決手段】

暗号化したいデータに冗長性を付加し、その長さよりも真に長い長さの鍵ストリームにより暗号化し、暗号文を生成する。復号化では冗長性の復元を確認することで通信中の改ざんを検出することができる。改ざんの成功確率が評価されており、ブロック暗号よりも高速で、並列度が高く、実装規模を小さくできる。

【選択図】 図 1

出 願 人 履 歴 情 報

識別番号 [000005108]

1. 変更年月日 1990年 8月31日

[変更理由] 新規登録

住 所 東京都千代田区神田駿河台4丁目6番地
氏 名 株式会社日立製作所